

Developing FLASH[®] LITE[™] 2.x and 3.x Applications

© 2008 Adobe Systems Incorporated. All rights reserved.

Developing Adobe® Flash® Lite™ 2.x and 3.x Applications

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.


The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, ActionScript, Creative Suite, Dreamweaver, Flash, Flash Lite, and Macromedia are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Macintosh is a trademark of Apple Inc., registered in the United States and other countries. Symbian and all Symbian based marks and logos are trademarks of Symbian Limited. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. SVG is a trademark of the World Wide Web Consortium; marks of the W3C are registered and held by its host institutions MIT, INRIA and Keio. ITC American Typewriter is a registered trademark of International Typeface Corporation. Arial is a trademark of The Monotype Corporation registered in the U.S. Patent and Trademark Office and certain other jurisdictions. Helvetica is a trademark of Heidelberger Druckmaschinen AG exclusively licensed through Linotype Library GmbH, and may be registered in certain jurisdictions. All other trademarks are the property of their respective owners.

 Sorenson Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

MPEG Layer-3 audio compression technology licensed by Fraunhofer IIS and Thomson Multimedia (<http://www.iis.fhg.de/amm/>).

Portions licensed from Nellymoser, Inc. (<http://www.nelly-moser.com>).

Adobe Flash 9.2 video is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are “Commercial Items,” as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Chapter 1: Flash Lite 2.x and 3.x overview

Flash Lite basics	1
Flash Lite 3.1 features	2
Flash Lite 3.0 features	7
Flash Lite 2.x features	10

Chapter 2: Getting started with Flash Lite

Hello World Flash Lite application	14
The Adobe Device Central emulator and device database	14
Workflow for authoring Flash Lite applications	15
About Flash Lite content types	16

Chapter 3: Tutorial: Creating a Flash Lite application

View the completed application	17
Create the application	18

Chapter 4: Creating interactivity and navigation

User interaction and supported keys	33
Default navigation	35
Key and button events	40

Chapter 5: Working with text and fonts

Text	50
Text input	51
Font rendering	58
Scrolling text	61

Chapter 6: Working with sound

Device sound	63
Native sound	67

Chapter 7: Working with video

Working with FLV files	71
Working with device video	74

Chapter 8: Developing Flash Lite applications for BREW

BREW basics	82
Setting up your system for BREW	85
Authoring Flash Lite files for BREW	87
Publishing Flash Lite files for BREW	89
Uploading files to BREW devices	97

Chapter 9: Testing Flash Lite content

Testing overview	102
Using the emulator	103

Chapter 10: Introduction to Flash Lite 2.x and 3.x ActionScript	
Supported, partially supported, and unsupported ActionScript classes and language elements	108
Unsupported and partially supported classes: details	155
Unsupported and partially supported ActionScript elements: details	171
Chapter 11: Warning and error messages	
Device error codes	174
Adobe Device Central emulator error and information messages	175
Index	181

Chapter 1: Flash Lite 2.x and 3.x overview

Adobe® Flash® Lite™ is a version of Flash® Player designed for devices. This documentation covers Macromedia® Flash® Lite™ 2.0 and Macromedia Flash Lite 2.1 software from Adobe, which are collectively called Flash Lite 2.x. It also covers Adobe Flash Lite 3.0 and 3.1, collectively called Flash Lite 3.x.

Flash Lite basics

About Flash Lite

Macromedia® Flash Lite 1.0 and Macromedia Flash Lite 1.1 are based on Macromedia Flash Player 4. Flash Lite 2.0 and 2.1 (collectively called 2.x) are based on Macromedia Flash Player 7 and support most—but not all—features in Flash Player 7. Flash Lite 2.x also includes features specific to mobile development that are not available in Flash Player 7. For example, in Flash Lite 2.x, you can load device-specific media types (images, sounds, video) that aren't natively supported by Flash Lite. Flash Lite 2.x also includes device integration features, such as the ability to make phone calls and send text messages.

Flash Lite 3.0 is based on Flash Player 7, and it introduces support for Flash Video (FLV), for browsing most Flash 8 content, as well as numerous performance improvements. A new security model that mirrors the model used in Flash Player 8 is also included in Flash Lite 3.0.

Flash Lite 3.1 is also based on Flash Player 7. It offers features to improve the web browsing experience for users of mobile devices. Many of these features were present in Flash® Player 8 but not supported in Flash Lite; they are now supported in Flash Lite 3.1. Flash Lite 3.1 introduces the ability to play back ActionScript™ 2.0 content when encountered on Flash 9 websites. However, Flash Lite 3.1 does not support processing and playback of ActionScript™ 3.0, which was introduced in Flash Player 9. See “[Flash Lite 3.1 features](#)” on page 2 for more information.

About components in Flash Lite

The components installed with Flash (for example, DataGrid and Accordion) were designed for use in Flash desktop applications. The memory requirements and processing power that they require typically prohibit their use in Flash Lite applications. Adobe recommends that you don't use the standard user interface components in your Flash Lite applications.

Optimizing content for performance and memory

To optimize your Flash content, you must pay attention to basic principles. For example, Flash developers have often had to avoid extremely complex artwork, excessive tweening, and overusing transparency.

Although earlier versions of Flash have resolved most of these performance issues for the desktop, Adobe Flash Lite developers still face additional challenges due to limitations of mobile devices: some devices perform better than others, sometimes dramatically, and because mobile authoring often requires publishing to many different devices, developers must sometimes author for the lowest common denominator.

Optimizing mobile content often requires making trade-offs. For example, one technique might look better, while another results in better performance. As you measure these trade-offs, you will be going back and forth repeatedly between testing in the emulator and testing on the target device.

In Flash 8, the emulator for Flash Lite 2.x was part of the Flash authoring environment. Beginning in Flash CS3 and Flash Lite 3.0 and continuing with Flash CS4 and Flash Lite 3.1, the emulator functionality is part of Adobe® Device Central. Device Central lets you emulate your Flash Lite projects on a variety of devices, and can emulate device display, memory use, and performance on specific devices. For complete information about using the emulator to optimize your Flash Lite content for mobile devices, see “Best practices for content on mobile devices” in the Device Central documentation.

Flash Lite 3.1 features

Improved web browsing

Flash Lite 3.1 offers a variety of features to improve the web browsing experience for users of mobile devices. Many of these features were present in Flash® Player 8 but not supported in Flash Lite; they are now supported in Flash Lite 3.1. These new features are described below.

Flash 9 browsing

Flash Lite 3.1 was released with support for Flash Player 8 and thus does not support ActionScript™ 3.0, which was introduced in Flash Player 9. However, many Flash 9 websites contain ActionScript 2.0 content, which could not be played back in Flash Lite 3.0. Flash Lite 3.1 introduces the ability to play back this content when encountered on Flash 9 websites. The Flash Lite 3.1 player publishes itself as a player capable of handling Flash 9 content. When websites that contain Flash 9 content are loaded, Flash Lite 3.1 evaluates the SWF file and plays the Flash 9 content that uses ActionScript 2.0 only. If the Flash 9 content uses ActionScript 3, Flash Lite 3.1 does not play it, but instead displays a user-friendly error icon (same as Flash Lite 3.0).

LocalConnection object

The LocalConnection class lets you develop one or more SWF files that can send instructions to each other without the use of `fscommand()` or JavaScript. LocalConnection objects can communicate only among SWF files that are running on the same client, but they can be running in different applications. LocalConnection objects can also be used to send and receive data within a single SWF file, but this capability is not widely used. The Flash Lite 3.1 implementation of the LocalConnection object supports communication among SWF files running in the same process (for example in the same browser window).

HTML text handling

Flash Lite 3.1 introduces support for several HTML text-handling tags that were not supported in Flash Lite 3.0. Flash Lite 3.1 includes support for the use of text as links, support for the `` tag, and support for handling cascading style sheets (CSS).

Text as links The `<a>` tag is used to create a hypertext link and is now supported in Flash Lite 3.1. See the Flash Lite documentation for complete information about the `<a>` tag’s attributes and usage.

Embedded external images Text fields in Flash Lite 3.1 support the `` tag. This tag can point to images stored in the local file system or on the network. The image can be in JPG, GIF, or PNG format, and will be rendered if the platform supports that image type. The text field itself can have different attributes, such as multiline and word wrap, which affect how the text is rendered. In addition to images, the `` tag can also point to a SWF file or MovieClip object. When embedded in the text field, these media are treated as children of the text field, and thus can be controlled and manipulated through ActionScript.

CSS support Text formatting with CSS styles was not supported in Flash Lite 3.0. Flash 8 content viewed in Flash Player on the desktop supports CSS-style formatting—either loaded as a separate CSS style sheet file or by adding

inline styles through ActionScript. The CSS feature allows developers to fine-tune the look and feel of their applications. In Flash Lite 3.1, this feature is supported.

Support for WMODE

Flash Lite 3.1 introduces support for the `wmode` parameter. Using this parameter in the context of the ActiveX plugin and a browser, you can manipulate the background of the HTML page that contains the Flash movie to control whether the background color or image shows through or not, making it possible to layer rich media content with DHTML content. The value of `wmode` can be `window` (the default), `opaque`, or `transparent`:

- `Window` plays the application in its own rectangular window on a web page. `Window` indicates that the application has no interaction with HTML layers and is always the topmost item.
- `Opaque` makes the application hide everything behind it on the page.
- `Transparent` makes the background of the HTML page show through all the transparent portions of the application and can slow animation performance.

Support for scale, salign, and align

Flash Lite 3.1 introduces support for the JavaScript parameters `scale`, `salign`, and `align` parameters, which control how rich media content gets scaled, positioned, and aligned when embedded in an HTML page. The `scale` parameter defines how the SWF file is placed in the browser window when width and height values are percentages. Its values can be `showall`, `noborder`, or `exactfit`. The `align` attribute determines how the SWF file is positioned within the browser window. Its values can be `L` (left), `R` (right), `T` (top), or `B` (bottom). The `salign` parameter specifies where a scaled SWF file is positioned in the area that the width and height settings define. Its values can be `TL` (top left), `TR` (top right), `BL` (bottom left), or `BR` (bottom right).

GetURL _target parameter

`_target` is an optional parameter of the ActionScript global `getURL` function that specifies the window or HTML frame that the document should load into. The `_target` parameter was not previously supported for Flash Lite applications because browsers on mobile devices have not supported multiple windows until recently. You can enter an empty string or choose from the following reserved target names:

`_self` specifies the current frame in the current window.

`_blank` specifies a new window.

`_parent` specifies the parent of the current frame.

`_top` specifies the top-level frame in the current window. If no `_target` is specified, then the default is `_self`. There must be frames in the browser window for all of the `_target` parameters to work as specified. If there are no frames in the browser, `_blank` works as expected—it opens the URL in a new blank window. However, the `_self`, `_parent`, and `_top` parameters open the URL in the same window.

SharedObject behavior changes

The rules for storage of local shared objects in Flash Lite 3.1 are now fully compatible with those of the Flash Player 8 (desktop). Before Flash Lite 3.1, the location of a shared object was derived from the name of the SWF file, so a shared object stored by `A.swf` could not be accessed by `B.swf`, no matter where these SWF files were located. In Flash Lite 3.1, the location of a shared object is derived from its URL instead of its name, so `/data/A.swf` located on a mobile device can access shared objects stored by `/B.swf`. Another key difference between Flash Lite 3.1 and previous versions is that in previous versions, if a SWF file was modified, even if its name remained the same, its shared objects were treated as different from the original ones. In Flash Lite 3.1, a SWF file's shared object is treated as the same as long as the URL/sandbox is the same, even if the SWF file itself has been modified or renamed. Further, prior to 3.1, two different

versions of a SWF file could not access each other's shared objects; in 3.1, they can. Flash Lite 3.1 builds on some important behavioral changes that were introduced in Flash Lite 3.0. Before Flash Lite 3.0, Flash Lite shared objects were available only to locally stored SWF files. SWF files playing back in a network-enabled browser could not use Flash Lite shared objects. In Flash Lite 3.0 and later, SWF files running in the network security sandbox can access and store shared objects. Also, starting in Flash Lite 3.0, all network SWF files could store and access local shared objects, but access was limited to a per-SWF basis. In Flash Lite 3.1, all SWF files originating from the same network domain can access each other's shared objects.

Optional parameters of `SharedObject.getLocal`

The `SharedObject` class has a `getLocal` method with two optional parameters.

- `secure`: a Boolean parameter (introduced in Flash 8) that you can use to distinguish between shared objects that originated from HTTP and HTTPS domains. Flash Lite 3.1 now supports the `secure` parameter.
- `localPath`: not supported in Flash Lite.

Image API

Flash Lite 3.1 supports APIs that application developers use to provide increased expressiveness by translating, rotating, scaling, and skewing images in mobile applications. The `flash.display` and `flash.geom` packages, first introduced in Flash 8, form the basis of this new functionality. The `flash.display` package supports the `BitmapData` class, which you use to create bitmaps of arbitrary sizes, and `flash.geom` supports classes and methods you use to perform pixel-level manipulations, including translation, rotation, scaling, and skewing. The *Adobe Flash Lite 2.x and 3.x ActionScript Language Reference* provides complete details about using these packages.

`flash.display.BitmapData` class

This class lets you create arbitrarily sized transparent or opaque bitmap images and manipulate them in various ways at runtime, which separates bitmap rendering operations from the Flash Lite player's internal display updating routines. You can attach `BitmapData` objects to `MovieClip` objects using the `MovieClip.attachBitmap()` method. The following `BitmapData` class methods are partially supported or not supported in Flash Lite 3.1. All other methods work as documented in the Flash Lite documentation.

Partially supported methods

The following `BitmapData` class methods are partially supported in Flash Lite 3.1.

Method	Description
<code>copyPixels</code>	Provides a fast routine to perform pixel manipulation between images with no stretching, rotation, or color effects. Flash Lite 3.1 does not support the <code>alphaBitmap</code> , <code>alphaPoint</code> , or <code>mergeAlpha</code> parameters of this method.
<code>draw</code>	Draws a source image or movie clip onto a destination image, using the Flash Lite player vector renderer. Flash Lite 3.1 does not support the <code>blendmode</code> parameter of this method.

Unsupported methods

The following `BitmapData` class methods are not supported in Flash Lite 3.1.

Method	Description
<code>applyFilter</code>	Takes a source image and a filter object and generates the filtered image. Flash Lite 3.1 does not support filters, so this method is not supported.
<code>generateFilterRect</code>	Determines the destination rectangle that the <code>applyFilter()</code> method call affects, given a <code>BitmapData</code> object, a source rectangle, and a filter object. Flash Lite 3.1 does not support filters, so this method is not supported.
<code>noise</code>	Fills an image with pixels representing random noise. Flash Lite 3.1 does not support this method.
<code>paletteMap</code>	Remaps the color channel values in an image that has up to four arrays of color palette data, one for each channel. Flash Lite 3.1 does not support this method.
<code>perlinNoise</code>	Creates a Perlin noise image. Flash Lite 3.1 does not support this method.
<code>pixelDissolve</code>	Performs a pixel dissolve either from a source image to a destination image, or by using the same image. Flash Lite 3.1 does not support this method.
<code>scroll</code>	Scrolls an image by a certain (x,y) pixel amount. Flash Lite 3.1 does not support this method.
<code>threshold</code>	Tests pixel values in an image against a specified threshold and sets pixels that pass the test to new color values. Flash Lite 3.1 does not support this method.

flash.geom.Point class

This class represents a location in a two-dimensional coordinate system, where *x* represents the horizontal axis and *y* represents the vertical axis. This class is fully supported in Flash Lite 3.1.

flash.geom.Matrix class

This class represents a matrix that determines how to map points from one coordinate space to another. By setting the properties of a `Matrix` object and applying it to a `MovieClip` or `BitmapData` object, you can perform various graphical transformations on the object. These transformation functions include translation, rotation, scaling, and skewing. Flash Lite 3.1 supports all of the methods of the `flash.geom.matrix` class except `createGradientBox`.

flash.geom.ColorTransform class

This class lets you mathematically adjust all of the color values in a movie clip. The color adjustment function of color transformation can be applied to all four channels: red, green, blue, and alpha transparency. This class is fully supported in Flash Lite 3.1.

flash.geom.Rectangle class

This class is used by the `BitmapData` object to manipulate a specific area in a bitmap. The class provides utility methods to manipulate one or more rectangles. This class is fully supported in Flash Lite 3.1.

flash.geom.Transform class

This class collects data about color transformations and coordinate manipulations that are attached to a `MovieClip` object. A `Transform` object is normally obtained by getting the value of the `transform` property from a `MovieClip` object. This class is fully supported in Flash Lite 3.1.

setTimeout/clearTimeout

The ActionScript 2.0 `setTimeout` and `clearTimeout` global functions were not supported in Flash Lite 3.0, but are supported in Flash Lite 3.1. The `setTimeout` function is used to run a specified function after a specified delay (in milliseconds). Note that using `setTimeout/clearTimeout` in conjunction with inline input text can cause problems for time-sensitive applications, such as online quizzes or games, because background processing of all ActionScript (including the timer) is paused when text is being input by the device user.

Streaming video seek functionality

Flash Lite 3.1 supports seek functionality in streaming Flash video (FLV). In Flash Lite 3.0, video playback included only minimal support for seeking. Seeking to a forward or backward position relative to the current head position was supported only for small HTTP videos. Flash Lite 3.1 supports the ability to seek within large FLV videos as well. Seeking backward is always supported, while seeking forward is supported with some limitations. Forward seek works only for the portion of the stream that has been downloaded. Attempting to seek to a position beyond the end of the downloaded stream does not result in a seek; rather, the video continues to play from the current head position. The seek functionality is used along with the `Netstream` class. `Netstream.seek()` is used to seek forward or backward in the HTTP video stream.

Streaming mp3 files

Flash Lite 3.1 supports the streaming of mp3 audio files using the ActionScript Sound object. In Flash Lite 3.0, mp3 files had to be completely downloaded before they could be decoded and played, and if the device could not allocate enough memory to process the entire file, then errors would result. Flash Lite 3.1 supports progressive sound streaming over both HTTP and local file system protocols, which means that mp3 files can be streamed from a web server or from the device's local file storage. Data in mp3 format is processed as it is downloaded, which optimizes memory consumption and allows devices to play back larger mp3 files than was possible in Flash Lite 3.0.

ActionScript extensions

The Extensions feature in Flash Lite 3.1 allows third-party developers to add features and ActionScript APIs to Flash Lite using a static or dynamically linked library. The DLL provides the implementation for the features, and Flash Lite allows the DLL to be used by calling ActionScript APIs. This makes it possible for Flash Lite developers to interact with APIs exposed by host devices by writing a DLL that bridges the capabilities of the device and of the Flash Player. These extensions can be statically or dynamically linked to the host implementation. For example, using this capability, developers can write applications that access the device's filing system, or use GPS data provided by the device, and so on.

Note: the ActionScript Extensions feature is available to third-party developers only if it has been enabled by the manufacturer of the target device.

Video enhancements

The following enhancements have been made to video functionality in Flash Lite 3.1.

H.264 video playback Like the desktop Flash Player version 9.2, Flash Lite 3.1 can play back Flash 7, 8, and 9 SWF files that use FLV by replacing FLV with H.264 on the server (Flash Media Server or HTTP).

Video postprocessing Flash Lite 3.1 supports video postprocessing (using the Video object's smoothing and deblocking properties) that result in a higher quality of display of video files on devices.

Flash Lite 3.0 features

Flash Lite 3.0 new features

Performance improvements Introduces a variety of player alterations to improve the player's performance on mobile devices and to improve the device user's experience when viewing and interacting with content. See "[Performance improvements](#)" on page 7 for more information.

Enhanced video capabilities Supports both Flash Video (FLV) and device video. See "[Enhanced video capability](#)" on page 7 for more information.

Web content browsability Supports the playback of most Flash 8 content on mobile devices with some loss of fidelity. See "[Web content browsability](#)" on page 8 for more information.

Local file security Provides a security scheme for Flash Lite that is consistent with the security model introduced in Flash Player 8. See "[Local file security](#)" on page 8 for more information.

Content Management Provides an API to read SWF metadata. Content developers can use metadata to segment content on devices and control some aspects of how content is handled. See "[Content management](#)" on page 9 for more information.

Performance improvements

Graphics rendering acceleration Renders text scrolling, vector translations, and animated vectors with opaque fills quickly and efficiently.

Run-time memory reduction "Lazy-loads" ActionScript™ classes at runtime, so that only the ActionScript required to render the first frame is loaded at startup. Other ActionScript classes are loaded as needed by the content.

Enhanced video capability

Flash Lite 2.x supports only device video, which means it relied on the device to decode and render video. It is limited by the video formats supported by a specific device, can't synchronize the video with the timeline, can't control the volume of the video clip, and can't rotate, skew, composite, or blend the device video with other media. (The video is always on top of any other Flash content.)

Flash Lite 3.0 adds support for Flash Video (FLV) using versions of the On2 and Sorenson codecs optimized for mobile devices. FLV video is rendered directly in the Flash Lite player rather than by the device, so you no longer need to be concerned about whether your target devices support a particular video format. Flash Lite 3.0 supports the playback of Flash Video using the following methods:

- Embedding video in SWF files
- Progressive downloading of video from local folders and web servers (HTTP)
- Streaming video from a Flash Media Server

See "[Working with video](#)" on page 71 for complete information about using FLV in Flash Lite 3.0.

Web content browsability

Flash Lite 3.0 supports the playback of most Flash 8 content on mobile devices with some loss of fidelity. Because of the high processor and memory requirements necessary to support features found in the Flash 8 Expressiveness feature set, many of these features are not supported in Flash Lite 3.0. These expressiveness features include bitmap caching, blend modes, bitmap filters, enhanced strokes, enhanced gradients, and 9-slice scaling. Whenever Flash Lite encounters any of these features in a SWF file, it renders these objects without the added expressiveness to maintain a usable mobile experience.

The following table describes the Flash 8 content and features that are supported, partially supported, or not supported.

Capability	Support	Details
Flash 8 basic HTML tags	Yes	The Flash Lite player can now recognize and play back most Flash 8 content, with some loss of fidelity.
Security enhancements	Yes	Flash Lite's security model is now consistent with the desktop player's security model. See "Local file security" on page 8 for details.
Flash Video	Yes	The Flash Lite player now supports Flash Video (FLV) using versions of the On2 and Sorenson codecs optimized for mobile devices. Also supports RTMP client implementation (one-way, not two-way real-time). See "Enhanced video capability" on page 7 for details.
HTML Base tag	Yes	Allows Flash Lite to behave in a manner similar to Flash Player on the desktop.
FlashType text	Partially supported	Complete support is not critical to displaying content. Text is rendered, but without full FlashType implementation.
Bitmap caching, effects, and enhancements	No	Not implemented because most require floating-point support or make excessive demands on processor or memory resources.
Focal gradients	No	Not implemented because they are processor-intensive and not critical to displaying content.

Local file security

In Flash Lite 2.x, local SWF files are allowed to interact with and load data from other SWF files, whether local or remote. This creates a security vulnerability because an untrusted local SWF file, downloaded (for example) as an e-mail attachment, can access private data stored on the device and send it back to the content author via an HTTP connection.

Flash Lite 3.0 introduces a new security model based on the Flash Player 8 security model. All SWF files are placed in a "sandbox" when they are loaded into the Flash Lite player. SWF files are assigned to different sandboxes according to their point of origin.

SWF files from the network are always placed in a remote sandbox that corresponds to their domain of origin. Network SWF files can read the data from their domain of origin only. They cannot access data from other domains unless those domains explicitly grant permission to the SWF file making the request. For example, a SWF file from a.com may read (using `XML.load`, for example) from the server at b.com if b.com has a policy file that permits access from a.com (or from all domains). A SWF file from a.com may cross-script a SWF file from b.com (calling an ActionScript™ method in the b.com SWF file, for example) if the b.com SWF file calls `System.security.allowDomain` to permit access from a.com.

SWF files from local origins (local file system) are placed into one of three sandboxes, as follows.

Local With Filesystem SWF files may read (using `XML.load`, for example) from files on local file systems, but they may not communicate with the network in any way.

Local Trusted SWF files may read from local files, interact with any server, and script any other SWF file. (In Flash Lite 2.1, all local files are effectively in this sandbox.)

Local With Networking Sandbox SWF files are allowed to communicate with other local-with-networking SWF files, and send data to servers (using `XML.send`, for example).

***Note:** This new security scheme affects the functionality of almost all ActionScript that involves data loading or cross-scripting, including extremely common functions like `getURL`, `loadMovie`, `loadMovieNum`, `loadVars`, `loadSound`, `XMLSocket.send`, and `XMLSocket.load`. For a comprehensive list of all API functionality that is affected by security, please see [Flash Player 8 Security-Related APIs \(PDF\)](#) on the [Flash Player Developer Center](#) website.*

See Flash 8 product Help for information about how the Flash 8 Player's security model works. When using this Help, note that the following differences exist between the Flash Lite 3.0 and Flash Player 8 security models:

- Trust management is done by the host application in Flash Lite.
- No trust configuration files are required in Flash Lite.
- SWF file developers, manufacturers, and carriers must make all security decisions in Flash Lite. There is no user-mediated trust, or Settings Manager. However, the host application may provide a mechanism by which users can make limited security decisions. For example, the host may allow users to designate a particular SWF file as trusted.
- Prohibited operations fail silently; no dialog box is displayed in Flash Lite.

Content management

Flash Lite 3.0 provides an API that allows a device's host software to read information from SWF metadata, which is standard information embedded in SWF files. If the host chooses to implement this functionality, then the metadata information can be used by Flash Lite content developers to organize and categorize content on the device.

The following categories of information can be included as metadata. Please note that some of these categories may be implemented differently (or not at all) by the manufacturer of the devices for which you are developing content.

Title	Description
Content Author	Name of content author.
Company Name	Company name of content author.
Title	Title of content.
Description	Description of the content.
Copyright Information	Relevant copyright details.
Content Version	Version number of content (NOT Flash version).
Content Type(s)	Content type(s) categories for content file. See the content type table that follows for content type lexicon.
Forward Lock	Specifies if the content file can be transferred to another device.
Supported Display Resolutions	List of resolutions supported by content.
Icon	Type and location of icon for content.
Flash Player Profile	Version of Flash/Flash Lite for which the content was published.

Title	Description
Storage Size	Storage size (size of <code>SharedObject</code>) required by the content. If size is unspecified, the content is assigned the default size set by the platform (generally 4KB).
Background Alpha	Preferred alpha value and blend mode for the background transparency feature.
Looping	Looping behavior: content either repeats indefinitely or stops after the last frame.
Default Quality	Quality level at which the content should be played.

The following content types can be used to segment content for devices and control some aspects of how the content is handled.

Content type	Description
Screensaver	Flash animation that is shown when a device-specified timeout occurs
Ringtone	Animation played when phone is receiving an incoming call
Background	Animated wallpaper that is displayed behind the idle/home screen
Game	Interactive game
Application	Standalone application
Presentation	Presentation converted to a Flash movie
Icon	Flash-based animated icon
Movie	Animated Flash movie
Skin	Skin or theme for the phone or a single application
Startup	Startup animation
Shutdown	Shutdown animation
Other	Flash content that does not fit into any of the above categories

Flash Lite 2.x features

About Flash Lite 2.x features

Flash Lite 2.0 and 2.1 (collectively called 2.x) are based on Flash Player 7 and support most of the features available in that version of Flash Player, including XML processing and ActionScript 2.0. Flash Lite 2.x also provides several features, not available in Flash Player 7, that are designed for mobile applications.

Flash Lite 2.1 new features

Flash Lite 2.1 adds support for inline text input. In previous versions of Flash Lite, input text fields used the device's generic input text dialog box. In Flash Lite 2.1, input text fields can be edited directly.

Note: Devices that support complex languages (for example, Hebrew, Arabic, Farsi, and some Asian languages) do not support inline text input. The functionality of input text fields on these devices will be the same for Flash Lite 2.1 as it is for Flash Lite 2.0.

Flash Lite 2.1 enables socket communication using the XMLSocket class on devices that support it. (For more information, see the *Flash Lite 2.x and 3.0 ActionScript Language Reference*.) For more information about input text fields, see “[Working with text and fonts](#)” on page 50.

Flash Lite 2.1 also adds tools to develop Flash Lite applications that run on devices using the Binary Runtime Environment for Wireless® (BREW®) platform made by QUALCOMM Incorporated. For more information about these tools, see “[Developing Flash Lite applications for BREW](#)” on page 82.

Flash Lite 2.0 new features

Flash Lite 2.0 includes the following new features:

- Support for ActionScript 2.0, which lets you use advanced programming techniques including classes, interfaces, and strict data typing
- Device video playback
- Local, persistent data storage (Flash Lite shared objects)
- Support for loading device-specific sound and image formats
- New system capabilities information
- Support for additional device keys, including QWERTY keyboard support and support for up to 11 soft keys
- Rich text formatting (partial support)
- Ability to control backlight duration and set custom focus rectangle color
- Synchronized device sound
- XML processing support

The following features in Flash Player 7 are not available in Flash Lite 2.0:

- Several ActionScript classes available in Flash Player 7 are unsupported or partially supported in Flash Lite 2.0. For more information about available ActionScript, see *Introduction to Flash Lite 2.0 ActionScript*.
- Socket communication using the XMLSocket class (available in Flash Lite 2.1)
- Support for communication with Flash Media Server (available in Flash Lite 3.0)
- Remote shared objects (local shared objects are partially supported)
- Native support for Flash Video (FLV) playback (available in Flash Lite 3.0)
- Support for Flash Application Protocol (the binary data communication protocol used by Flash Remoting)
- Cascading Style Sheet (CSS) formatting with text fields
- Masking with device fonts
- Bitmap smoothing while rendering at high-quality

Flash Lite 2.0 ActionScript

Flash Lite 2.0 ActionScript is the scripting language used in Flash Lite 2.0 and Flash Lite 2.1 applications. It shares some, but not all, of the ActionScript used in Flash Player 7. Flash Lite 2.0 also includes several ActionScript additions and extensions that let you, for example, get information about the device, make phone calls, or control the backlight duration.

You can use ActionScript 2.0 or ActionScript 1.0 syntax when you develop applications for Flash Lite 2.0. ActionScript 2.0 provides authoring support for classes, interfaces, and strict data typing. Using ActionScript 2.0 syntax lets the ActionScript compiler provide better debugging information, and also encourages better program design.

For more information about learning Flash Lite 2.0 ActionScript, see the following books and topics:

- *Introduction to Flash Lite 2.x and 3.0 ActionScript*
- *Flash Lite 2.x and 3.0 ActionScript Language Reference*
- The “Learning ActionScript 2.0” topic in *Using Flash*

Device video playback

Flash Lite 2.0 can play video in any format that’s supported natively by the target device. For example, some devices record and playback video in the 3GP video format. Others support AVI or MPEG video. During playback, Flash Lite passes the original video data to the device to decode and render the data directly to the screen. You can incorporate video in your application in any of the following ways:

- Bundle the original video data in the SWF file.
- Load an external video file from the device’s memory card, or over the network.

To control video playback in Flash Lite 2.0 you use the ActionScript Video object. First available in Flash Player 6, the Video object in Flash Lite 2.0 has additional methods for controlling video, as the `Video.play()` and `Video.pause()` methods. You can also use the `System.capabilities.videoMIMETypes` array to determine what video formats a device supports.

For more information about the Video object and using video in Flash Lite, see “[Working with video](#)” on page 71.

Loading device-specific sound and image formats

In Flash Lite 2.0, you can load any image or sound file that’s in a format supported by the device. To load external images, you use the `loadMovie()` global function, or the `MovieClip.loadMovie()` method. For example, if the device supports the PNG file format, then you could use the following code to load a PNG file from a web server into the movie clip instance `image_mc`:

```
image_mc.loadMovie("http://www.adobe.com/images/mobile.png");
```

To load external sounds, you use the `Sound.loadSound()` method. In Flash Lite 2.0, you can use this method to load any sound format that the device supports (for example, MIDI or SMAF). External device sounds must fully load into memory before they can play.

As in Flash Lite 1.x, in Flash Lite 2.0 you can also play device sound that’s bundled in the SWF file. For more information, see “[Using bundled device sound](#)” on page 63.

For more information about loading external images and sounds, see the following topic:

See also

“[Playing external device sounds](#)” on page 66

Flash Lite shared objects

Flash Lite shared objects let you save data persistently to the user’s device. For example, you might use a shared object to save information between application sessions, such as user preferences or game scores. You use the `SharedObject` class to read and write Flash Lite shared objects. For more information about using Flash Lite shared objects, see the `SharedObject` class in the *Flash Lite 2.x and 3.0 ActionScript Language Reference*.

Note: The Flash Lite 2.0 implementation of shared objects does not allow multiple SWF files to share the same data. Also, none of the Flash Lite versions supports remote shared objects with Flash Media Server.

Synchronized device sound

In previous versions of Flash Lite you could only synchronize native Flash sound to animation in the timeline. But this was not possible with device sounds, which are played directly by the device, rather than natively by Flash Lite. In Flash Lite 2.0, you can synchronize device sound with the timeline using the new `_forceframerate` property. When this property is set to `true`, Flash Lite drops frames as necessary from animation to maintain the frame rate specified in the SWF file. For more information, see “[Synchronizing device sounds with animation](#)” on page 67 and the `forceframerate` property in the *Flash Lite 2.x and 3.0 ActionScript Language Reference*.

Text features

The following features related to text handling are new in Flash Lite 2.0:

- All text in Flash Lite 2.0 is Unicode-based.
- Flash Lite 2.0 provides partial support for HTML formatting and the `TextFormat` ActionScript class.

For more information about working with text fields in Flash Lite 2.x, see “[Working with text and fonts](#)” on page 50.

Additional key support

Flash Lite 2.0 provides additional support for device keys, including support for QWERTY keyboards, and up to 12 soft keys (including the standard Left and Right soft keys).

Chapter 2: Getting started with Flash Lite

Hello World Flash Lite application

This simple tutorial introduces you to the mobile authoring and testing features in Adobe Flash CS4 Professional, as well as the general workflow for creating content using Adobe Flash Lite. In this section, you create a simple Flash Lite application and test it in the Adobe Device Central CS4 emulator. For a more complete sample application, see [“Tutorial: Creating a Flash Lite application”](#) on page 17.

For the purposes of this tutorial, assume that you’re developing content for the Flash Lite stand-alone player. The procedure targets a generic device.

First, decide which devices and Flash Lite content type you are targeting.

Configure and create a simple Flash Lite application

- 1 Start Flash.
- 2 On the main Flash screen, select Create New > Flash File (Mobile). Flash opens Adobe Device Central and displays the New Document tab.
- 3 On the New Document tab, select Flash Lite 2.0 in the Player Version box, Adobe ActionScript™ 2.0 in the ActionScript Version box, and Standalone Player in the Content Type box.
- 4 Click Custom Size for All Selected Devices at the bottom of the screen. This allows you to create content for the stand-alone Flash Lite player.
- 5 Click Create. You are returned to Flash, which creates a new document with preset publish settings and (when you specify a device) the correct size for the device you selected.
- 6 In the Tools panel, select the Text tool and drag to create a text box on the Stage.
Type **Hello, world!** (or other text) in the text box.
- 7 Select Control > Test Movie to export your application to Adobe Device Central and view your application in the Adobe Device Central emulator.

***Note:** During testing in Device Central, you can change the device and content type to see your application on a different platform. To do this, double-click a device in the library panel and select a new content type from Content Type. When you return to Flash, Flash remembers the settings you last used in the emulator.*

- 8 To return to Flash, select File > Return to Flash.

The Adobe Device Central emulator and device database

The Adobe Device Central emulator lets you test your content as it will run and appear on an actual device. You can select a different test device or content type and view detailed information about your application.

Device Central contains an extensive database of mobile devices from which you can select to create your own test environment. See the Device Central documentation for more information.

Workflow for authoring Flash Lite applications

Creating Flash Lite content is an iterative process that involves the following steps:

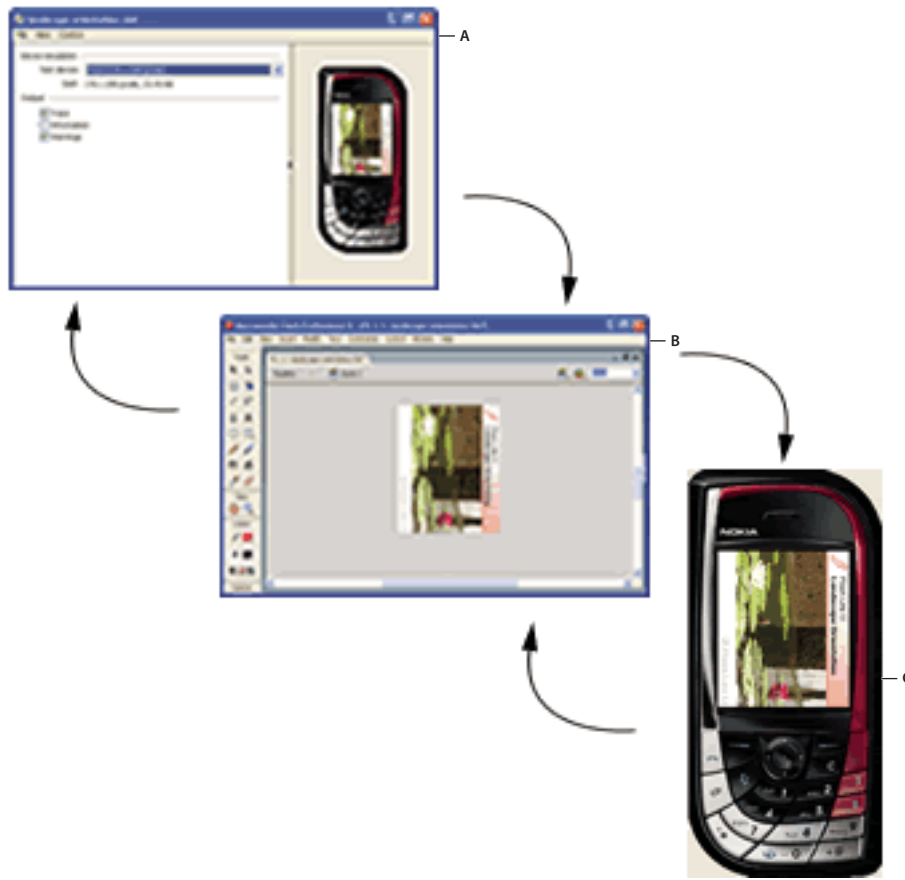
Identify your target devices and Flash Lite content type Different devices have different screen sizes, support different audio formats, and have different screen color depths, among other factors. These factors may influence your application's design or implementation.

In addition, different devices support different Flash Lite content types, such as screen savers, stand-alone applications, or animated ring tones. The content type for which you are developing also determines the features that are available to your application. For more information, see "[About Flash Lite content types](#)" on page 16.

Create your application in Flash and test in Adobe Device Central Adobe Flash CS4 Professional includes an emulator on Adobe Device Central that lets you test your application without having to transfer it to a device. You use the emulator to refine your application design and fix any problems before you test it on a mobile device.

Test the application on your target device or devices This step is important because the Adobe Device Central emulator doesn't emulate all aspects of the target device, such as its processor speed, color depth, or network latency. For example, an animation that runs smoothly on the emulator might not run as quickly on the device because of its slower processor speed, or a color gradient that appears smooth in the emulator may appear banded when viewed on the actual device. After you test your application on a device, you may find that you need to refine the application's design in the Flash authoring tool.

The following illustration shows the iterative development and testing process:



A. Editing the FLA file in Adobe Flash B. Testing in the Adobe Device Central emulator C. Testing on a device

About Flash Lite content types

Before you start developing a Flash Lite application, you need to know the following:

- The device or devices on which the content will run (*target devices*). The Flash Lite player is installed on a variety of devices. For a list of devices that have Flash Lite installed, see the Supported Devices page on the Adobe website at www.adobe.com/go/mobile_supported_devices/.
- The Flash Lite content types that the target devices support. Each Flash Lite installation supports one or more application modes (*content types*). For example, some devices use Flash Lite to enable Flash-based screen savers or animated ring tones. Other devices use Flash Lite to render Flash content that is embedded in mobile web pages. Not all content types support all Flash Lite features.

Each Flash Lite content type, paired with a specific device, defines a specific set of Flash Lite features that are available to your application. For example, a Flash application that is running as a screensaver is not typically allowed to make network connections or download data.

The Flash Lite testing features in Adobe Device Central let you test against multiple devices and different Flash Lite content types. This ability lets you determine if your application uses features that aren't available for the type of content that you are developing. For more information about selecting target devices and content types, see “[Testing Flash Lite content](#)” on page 102 in *Developing Flash Lite 2.x and 3.x Applications*.

Chapter 3: Tutorial: Creating a Flash Lite application

In this tutorial, you'll develop an Adobe Flash Lite application that promotes a fictional restaurant called Café Townsend. Users can view a list of specials at the restaurant, watch a video about the chef, and call the restaurant to make reservations. Users also set their preferred location for making reservations using a shared object, which stores their preference between application sessions.

View the completed application

There are two versions of the completed application: a version that uses a 3GP device video file, and a version that uses an FLV file. You can view each version in the Adobe Device Central emulator. You can also transfer the completed applications to the stand-alone Flash Lite player installed on a mobile device.

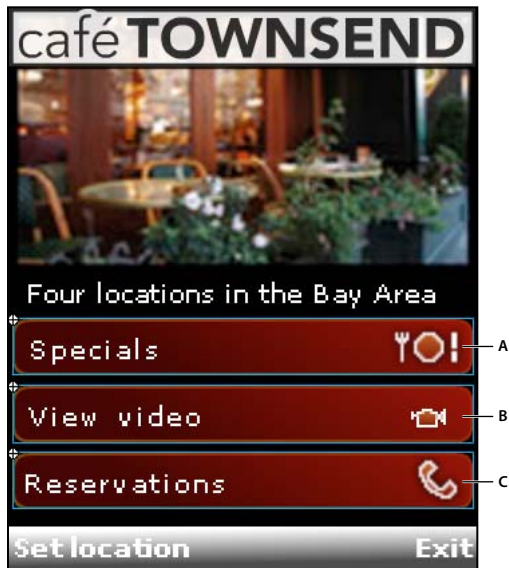
Device Central does not support every Adobe ActionScript command. For example, Device Central does not support `fscommand2("Quit")` or `getURL()`. Device Central does send messages to the Output window to verify that the commands were called. To view the messages, choose Window > Flash Output (Device Central CS4) or View > Flash Output > Show (Device Central CS3). For more information, see “Flash Lite features not supported by the emulator” on page 106.

- 1 Download and decompress the tutorial files. Browse to www.adobe.com/go/learn_flt_samples_and_tutorials. Download the ZIP file for your Flash Lite version.
- 2 In Flash, choose File > Open, and browse to the file named `cafe_tutorial_complete fla` or `cafe_tutorial_FLV_complete fla` in the Tutorial folder.
- 3 Select Control > Test Movie to test the application in Device Central.
- 4 To interact with the application, do the following:
 - On the homescreen, click the Down key to select the Specials menu item. Click the Select key to view the Specials screen.
 - On the Specials screen, click the Right soft key (Next) on the emulator to view the image and description for each special. Click the Left soft key (Home) to return to the homescreen.
 - Select the View Video menu item to watch the video. Click the Left soft key (Home) to return to the homescreen. You can also click the Right soft key (Pause) to pause the video.
 - On the homescreen, click the Left soft key (Set Location) to go to the screen where you set your location. Select a preferred location for reservations, and then click the Left soft key (Save) to set the location and return to the homescreen (or click Cancel).
 - On the homescreen, select the Reservations menu item to start a phone call to the restaurant. In the Output window, confirm that the phone number matches the location you set on the set location screen.

Create the application

Create the menu for the homescreen

- 1 In Flash, choose File > Open, and browse to the `cafe_tutorial_start.fla` file located in the Tutorials folder.
- 2 Choose File > Save As, and save the file as `cafe_tutorial.fla`.
- 3 In the Timeline (Window > Timeline), select Frame 1 in the Main menu layer.
- 4 Open the Library panel (Window > Library), and drag an instance of the Specials button to the Stage. Do the following:
 - a Position the Specials button beneath the text field that introduces the restaurant.
 - b With the Specials button selected, in the Property inspector, enter **specials_btn** in the Instance Name box.
- 5 Drag an instance of the video button from the Library panel to the Stage. Do the following:
 - Position the video button below the Specials button.
 - With the video button selected, in the Property inspector, enter **video_btn** in the Instance Name box.
- 6 Drag an instance of the Reservations button from the Library panel to the Stage. Do the following:
 - Position the Reservations button below the video button.
 - With the Reservations button selected, in the Property inspector, enter **reservations_btn** in the Instance Name box.
- 7 The Stage should look something like the following:



A. `specials_btn` B. `video_btn` C. `reservations_btn`

- 8 In the Timeline, select Frame 1 in the layer named ActionScript. Open the Actions panel (Window > Actions) and enter the following code:

```
stop();
_focusrect = false;
fscommand2("SetSoftKeys", "Set Location", "Exit");
fscommand2("Fullscreen", "true");
```

This code does the following:

- Stops the playhead at this frame.
- Disables the focus rectangle that Flash Lite draws by default around the button or input text field with the current focus (see “[About the focus rectangle](#)” on page 37).
- Registers the soft keys for your application to use.
- Displays the application full screen.

When Flash Lite is in full-screen mode, the labels you specify in the `SetSoftKeys` command are not visible. For this reason, you must add custom soft key labels to the Stage.

9 Add the following code on Frame 1 to handle button events for the menu buttons, and for selection focus:

```
// Set initial focus when the application
// starts and also upon returning to the main
// screen from another screen.
if (selectedItem == null) {
    Selection.setFocus (specials_btn);
} else {
    Selection.setFocus (selectedItem);
}
// Assign onPress event handlers to each menu button,
// and set selectedItem variable to selected button
// object:
specials_btn.onPress = function () {
    gotoAndStop ("specials");
    selectedItem = this;
};
video_btn.onPress = function () {
    gotoAndStop ("video");
    selectedItem = this;
};
reservations_btn.onPress = function () {
    if (location_so.data.phoneNumber == undefined) {
        // User hasn't specified location so
        // go to "set location" screen:
        gotoAndStop ("locations");
    }
    else {
        // Call number in shared object:
        var phoneNum = location_so.data.phoneNumber;
        getURL ("tel:" + phoneNum);
    }
    selectedItem = this;
};
```

The `onPress` event handlers assigned to the buttons named `specials_btn` and `video_btn` send the playhead to frames labeled, “specials” and “video.”

When the user selects the Reservations option, the `onPress` handler dials the phone number specified in the `location_so` shared object. (Later in this procedure, you’ll create code to create the shared object.) If the user hasn’t yet specified a location to call for reservations, the application sends the playhead to the frame labeled “locations,” where the user selects the preferred location for making reservations.

10 Add the following code to Frame 1 to create a key listener for the Left and Right soft keys:

```

Key.removeListener(myListener);
var myListener:Object = new Object();
myListener.onKeyDown = function() {
    var keyCode = Key.getCode();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event:
        gotoAndStop("locations");
    } else if (keyCode == ExtendedKey.SOFT2) {
        // Handle right soft key event:
        fscommand2("Quit");
    }
};
Key.addListener(myListener);

```

This code uses a key listener object to handle Right and Left soft key events. When the user presses the Left soft key, the playhead is sent to the frame labeled “locations.” The Right soft key closes the application.

For more information about using event listeners, see [“Handle keypress events through a key listener”](#) on page 46.

11 Add code to initialize the shared object that saves the preferred location for making reservations:

```

// Define Shared Object listener function:
function so_listener (the_so:SharedObject) {
    if (the_so.getSize () == 0) {
        // The shared object doesn't exist, so the user
        // hasn't set a preference yet.
    }
    SharedObject.removeListener ("location");
}
// Create shared object:
location_so = SharedObject.getLocal ("location");
// Add SharedObject listener object:
SharedObject.addListener ("location", this, "so_listener");

```

12 Choose File > Save.

13 To test your work in Device Central, select Control > Test Movie.

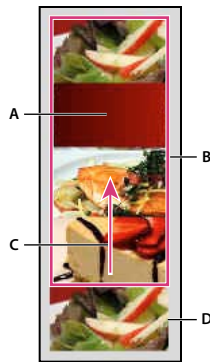
At this point, you can select a menu item by giving the corresponding button focus, and pressing the select key (or the Enter key on your computer keyboard). Open the Output window in Device Central to view feedback about unsupported commands.

Create the Specials screen

Create the animation that transitions between images of each special. Add user interface elements and ActionScript to let the user navigate between the images. Add ActionScript to display the name and description of each special below the image.

Create the image animation

To create the animation, use a prebuilt movie clip that contains images of all of the specials arranged in a vertical column. Use a masking layer to make only one of the images visible. A tween animation moves the images through the mask. The last image in the movie clip is a duplicate of the first image, so that the animation sequence can return to its initial state after the user views the final image.

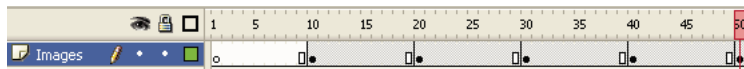


A. Masking layer object B. Masked movie clip of images C. Tween direction D. Duplicated image

- 1 Open the file `cafe_tutorial.fla` that you completed in the section “[Create the menu for the homescreen](#)” on page 18.
- 2 In the Timeline, select the keyframe in Frame 10 in the layer named **Specials Photos**.
- 3 Open the Library panel, and drag the **Specials** movie clip symbol to the Stage.
- 4 With the **Specials** movie clip selected, set the *x* and *y* coordinates to **0** in the Property inspector.

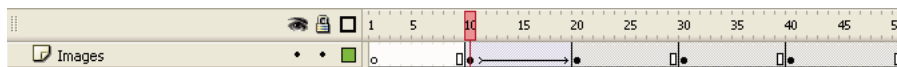
This aligns the upper-left corner of the **Specials** movie clip with the upper-left corner of the Stage.

- 5 In the **Specials Photos** layer, select Frames 20, 30, 40, and 50 and press F6 to insert keyframes. Select frame 51, and press F7 to insert a blank keyframe. The Timeline should look like the following image:



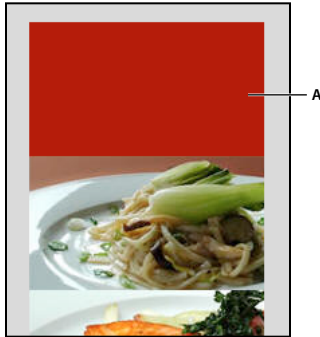
- 6 Select the keyframe in Frame 20, select the **Specials** movie clip, and set its *y* coordinate to **-100** in the Property inspector.
- 7 Select the keyframe in Frame 30 in the Timeline, select the **Specials** movie clip, and set its *y* coordinate to **-200** in the Property inspector.
- 8 Select the keyframe in Frame 40, select the **Specials** movie clip, and set its *y* coordinate to **-300** in the Property inspector.
- 9 Select the keyframe in Frame 50, select the **Specials** movie clip, and set its *y* coordinate to **-400** in the Property inspector.
- 10 Select the keyframe in Frame 10, and select **Motion** from the Tween pop-up menu in the Property inspector.

This tweens the **Images** movie clip’s position between the keyframes in Frames 10 and 20.



- 11 To create transitions between the other images, repeat step 11 for the keyframes located in Frames 20, 30, and 40.
- 12 To create the mask layer, select the **Specials photos** layer in the Timeline, and then select **Insert > Timeline > Layer** (or click **Insert Layer** in the Timeline).
- 13 Double-click the name of the new layer and rename it **Mask**.
- 14 Select Frame 10 of the **Mask** layer and choose **Insert > Timeline > Keyframe**.
- 15 Select the **Rectangle** tool in the Tools palette and create a rectangle over the first (uppermost) image in the **Images** movie clip.

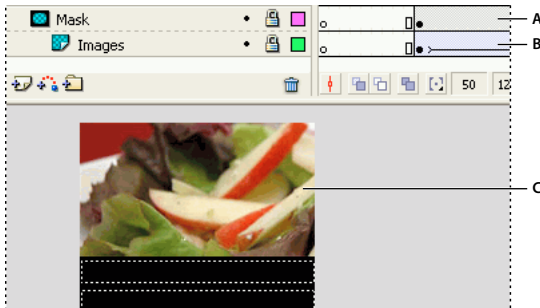
It doesn't matter what fill color you use for the rectangle, but it must be completely opaque.



Masking rectangle

- 16 To make sure the rectangle covers the entire image area, double-click the rectangle to select it, and then use the Property inspector to set its *x* and *y* coordinates both to **0**, its width to **176**, and its height to **100**.
- 17 Right-click (Windows®) or Control-click (Macintosh®) the Image Mask layer in the Timeline, and select Mask from the context menu.

The layer is converted to a mask layer, indicated by a mask layer icon. The layer immediately below it is linked to the mask layer, and its contents show through the filled area on the mask. For more information about working with mask layers in Flash, see the mask layers sections in *Using Flash*.



A. Mask layer B. Masked layer C. Masked region on the Stage

- 18 Save your changes (File > Save).

If you test the application now, the specials animation does not play. In “[Add navigation and text to the Specials screen](#)” on page 22, you add ActionScript that stops the animation at each keyframe. You also add user interface elements that let the user navigate between images.

Add navigation and text to the Specials screen

- 1 In Flash, open the `cafe_tutorial.fla` file.
- 2 In the Timeline, select Frame 10 in the Text layer.
- 3 In the Tools palette, select the Text tool and create a text field below the first masked-specials image.

This text field displays the name of the special whose image is shown on the screen.



Text field to display name of special

- 4 With the text field selected on the Stage, do the following in the Property inspector:
 - Select Dynamic Text from the Text Type pop-up menu.
 - Type **title_txt** in the Instance Name box.
 - Select the Italics text style option.
 - Set the font size to 10.
 - Set the font color to white.
 - Select Use Device Fonts from the Font Rendering Method pop-up menu.
- 5 Create another text field below the first text field to display a short description of the specials that the user is viewing.
- 6 Using the Selection tool, resize the new text field so that it's about three times as tall as the text field above it.



Text field to display description of special

7 With the text field selected on the Stage, do the following in the Property inspector:

- Select Dynamic Text from the Text Type pop-up menu.
- Type **description_txt** in the Instance Name text box.
- Select Multiline from the Line Type pop-up menu.
- Set the font size to 10.
- Make sure that Italic is not selected.
- Set the font color to white.
- Select Use Device Fonts from the Font Rendering Method pop-up menu.

8 In the Timeline, select the keyframe in Frame 10 in the ActionScript layer.

9 Open the Actions panel and add the following code:

```
stop();
fscommand2("SetSoftKeys", "Home", "Next");
title_txt.text = "Summer Salad";
description_txt.text = "Butter lettuce with apples, blood orange segments, gorgonzola, and
raspberry vinaigrette.";
```

This code displays the name description of the first special in the two dynamic text fields. It also stops the playhead on the current frame, and registers the soft keys.

10 In the ActionScript layer, select the keyframe in Frame 20 and enter the following code in the Actions panel:

```
stop();
title_txt.text = "Chinese Noodle Salad";
description_txt.text = "Rice noodles with garlic sauce, shitake mushrooms, scallions, and
bok choy.";
```

11 In the ActionScript layer, select the keyframe in Frame 30 and enter the following code in the Actions panel:

```
stop();
title_txt.text = "Seared Salmon";
description_txt.text = "Filet of wild salmon with caramelized onions, new potatoes, and
caper and tomato salsa.";
```

12 In the ActionScript layer, select the keyframe in Frame 40 and enter the following code in the Actions panel:

```
stop();
title_txt.text = "New York Cheesecake";
description_txt.text = "Creamy traditional cheesecake served with chocolate sauce and
strawberries.";
```

13 In the ActionScript layer, select the keyframe in Frame 50 and enter the following code in the Actions panel:

```
gotoAndStop("specials");
```

This code returns the playhead to the beginning of the animation sequence. The first and last images in the animation sequence are the same, which creates the illusion of a continuous animation.

14 In the Timeline, select Frame 10 in the layer named ActionScript. Open the Actions panel and enter the following code:

```
Key.removeListener (myListener);
var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        gotoAndPlay ("home");
    }
    else if (keyCode == ExtendedKey.SOFT2) {
        // Handle right soft key event
        play ();
        description_txt.text = "";
        title_txt.text = "";
    }
};
Key.addListener (myListener);
```

The Left soft key sends the playhead to the main application screen, and the Right soft key advances the image animation to the next image in the sequence.

For more information about using event listeners, see [“Handle keypress events through a key listener”](#) on page 46.

15 Choose File > Save.

16 Select Control > Test Movie to test the application in Device Central.

Select the Specials button. The Specials animation and text should work correctly.

Create the video screen

In this section, you add video to the application. You also add ActionScript code that lets the user control playback (play, pause) with the Right soft key.

Flash Lite supports two types of video: device video and FLV files (Flash Video). Flash Lite 2.x supports device video. Flash Lite 3.x supports device video and FLV files. To complete this tutorial, you must choose to use either device video or an FLV file.

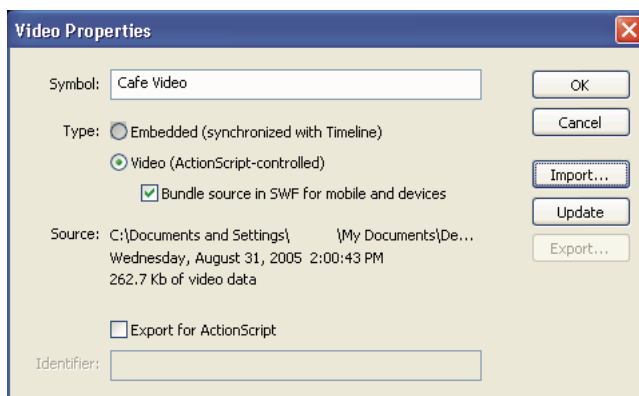
Note: For more information about video support, see [“Working with video”](#) on page 71.

Create the video screen with device video (Flash Lite 2.x or Flash Lite 3.x)

- 1 In Flash, open the `cafe_tutorial.fla` file you completed in “[Create the Specials screen](#)” on page 20.
- 2 Open the Library options menu located in the upper-right corner of the Library panel and select New Video.
- 3 In the Video Properties dialog box, do the following:
 - a Enter a name for the video symbol in the Symbol box (**Cafe video**, for example).
 - b Select Bundle source in SWF for mobile and devices. Click Import.
 - c Browse to the `cafe_townsend_chef.3gp` file in the Tutorials folder. Select the file and click Open.

If you don’t see the video file (or if you can see it but can’t select it), select All Files (*.*) from the Files Of Type pop-up menu (Windows), or All Files from the Enable pop-up menu (Macintosh).

- 4 Make sure the Video Properties dialog box appears as follows, and click OK:



A new video symbol appears in the Library panel.

- 5 Select frame 51 in the Video layer of the Timeline. Drag the Cafe video symbol from the Library panel to the Stage.
- 6 In the Property inspector, enter **cafeVideo** in the Instance Name box. Set the *x* position to **0**, the *y* position to **30**, the Width to **176**, and the Height to **144**.



- 7 In the Timeline, select Frame 51 of the ActionScript layer. Enter, or copy and paste, the following code in to the Actions panel:

```

// Stop timeline, register soft keys, and start video.
stop ();
fscommand2 ("SetSoftKeys", "Home", "Pause");
cafeVideo.play ();
var playing:Boolean = true;
// Soft key event handler code:
Key.removeListener (myListener);
var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        gotoAndPlay ("home");
    }
    else if (keyCode == ExtendedKey.SOFT2) {
        if (playing) {
            // If video is playing, pause it,
            // set status variable (playing) to false,
            // and set right soft key label to 'Play'.
            cafeVideo.pause ();
            playing = false;
            rightSoftKeyLabel.text = "Play";
        }
        else {
            // If video is paused, resume its playback,
            // set status variable (playing) to true,
            // and set right soft key label to 'Pause'.
            cafeVideo.resume ();
            playing = true;
            rightSoftKeyLabel.text = "Pause";
        }
    }
};
// Register listener object:
Key.addListener (myListener);
// Video status handler code.
// Device Central does not support the onStatus handler.
cafeVideo.onStatus = function (infoObject:Object) {
    var code = infoObject.code;
    if (code == "completed") {
        // If video has finished playing, set playing=false,
        // and set right soft key label to "Replay":
        playing = false;
        rightSoftKeyLabel.text = "Replay";
    }
};

```

8 Choose File > Save.

9 Choose Test > Control Movie to test the application in Device Central.

Select View Video on the homescreen to view the video. Press the Right soft key to pause the video. Press the Right soft key again to resume playback. On a device, you can press the Right soft key to play the video again after playback completes. However, this feature (the video.onStatus handler) is not supported in Device Central.

Create the video screen with an FLV file (Flash Lite 3.x)

1 In Flash, open the cafe_tutorial.fla file you completed in “[Create the Specials screen](#)” on page 20.

- 2 Locate the `cafe_townsend_chef.flv` file in the Tutorials folder. Copy this file to the same folder as the `cafe_tutorial fla` file.
- 3 Open the Library options menu located in the upper-right corner of the Library panel and select New Video.
- 4 In the Video Properties dialog box, do the following:
 - a Enter a name for the video symbol in the Symbol box (**Cafe video**, for example).
 - b Select Video (ActionScript-Controlled). Click OK.
- 5 Select Frame 51 in the Video layer of the Timeline. Drag the Cafe video symbol from the Library panel to the Stage.
- 6 In the Property inspector, enter **cafeVideo** in the Instance Name box. Set the *x* position to **0**, the *y* position to **45**, the Width to **176**, and the Height to **144**.



- 7 In the Timeline, select Frame 51 of the ActionScript layer. Enter, or copy and paste, the following code in to the Actions panel:

```
// Frame actions
stop();
fscommand2 ("SetSoftKeys", "Home", "Pause");
var videoIsPlaying:Boolean;
// Use the NetConnection and NetStream classes to play FLV files
var nc:NetConnection = new NetConnection();
var ns:NetStream;
/* When a change in connection status occurs,
Flash Lite triggers the onStatus handler and passes it an
object with information about the change.
Use the information object to run code when
connection events occur.
*/
nc.onStatus = function(info) {
    // Run the following code if the connection is successful.
    if(info.code == "NetConnection.Connect.Success"){
        // Create a NetStream object and pass it the NetConnection object
        ns = new NetStream(nc);
        // Attach the NetStream object to the video object on Stage.
        cafeVideo.attachVideo(ns);
        // Play the file "cafe_townsend_chef.flv".
        ns.play("cafe_townsend_chef.flv");
        videoIsPlaying = true;
    }
}
// Create a connection.
// To connect to a local file or to a file on a web server, pass null.
nc.connect(null);
/* When a change in stream status occurs,
```


Flash Lite triggers the onStatus handler and passes it an object with information about the change. Use the information object to run code when stream changes occur.

```
*/
ns.onStatus = function(info){
    if(info.code == "NetStream.Play.Stop"){
        videoIsPlaying = false;
        rightSoftKeyLabel.text = "Play";
    }
};
// Capture and handle soft key events.
Key.removeListener (myListener);
var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    // Run this code if the user presses the left soft key.
    if (keyCode == ExtendedKey.SOFT1) {
        // Close the video.
        ns.close();
        // Go to the home screen.
        gotoAndPlay ("home");
    }
    // Run this code if the user presses the right soft key
    else if (keyCode == ExtendedKey.SOFT2) {
        // If the video is playing, run this code
        if (videoIsPlaying){
            // Pause or restart the video (this method does both).
            ns.pause();
            // Switch the label text.
            if (rightSoftKeyLabel.text == "Play"){
                rightSoftKeyLabel.text = "Pause";
            }
            else if (rightSoftKeyLabel.text == "Pause"){
                rightSoftKeyLabel.text = "Play";
            }
        }
        // If the video has finished playing, run this code.
    } else {
        // Replay the video.
        ns.play("cafe_townsend_chef.flv");
        rightSoftKeyLabel.text = "Pause";
        videoIsPlaying = true;
    }
}
};
Key.addListener (myListener);
```

8 Choose File > Save.

9 Choose Test > Control Movie to test the application in Device Central.

Select View Video on the homescreen to view the video. Press the Right soft key to pause the video. Press the Right soft key again to resume playback. When the video has finished playing, press the Right soft key to play it again.

Create the set location screen

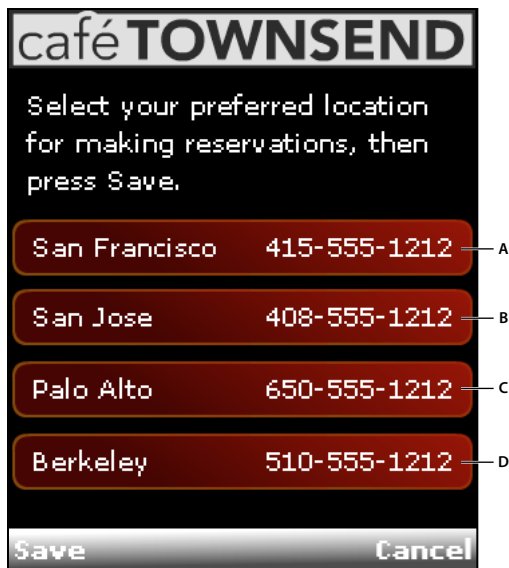
In this section, create a new screen that lets the user select which of the four restaurant locations they want to call for reservations.

Note: To see the number dialed, choose *Window > Flash Output in Device Central CS4* or *View > Flash Output > Show in Device Central CS3*.

The first time the user starts the application and selects Reservations, the application takes them to the set location screen. Then, when the user selects Reservations, the application dials the selected location. The application uses a shared object to save the location between sessions.

- 1 In Flash, open the `cafe_tutorial.fla` file you completed in “[Create the video screen](#)” on page 25.
- 2 In the Timeline, select the keyframe on Frame 66 of the layer named Options Menu.
- 3 Drag the `location_SF` button from the library to the Stage. In the Property inspector, enter **sf_btn** in the Instance Name box.
- 4 Drag the `location_SJ` button from the library to the Stage. In the Property inspector, enter **sj_btn** in the Instance Name box.
- 5 Drag the `location_PA` button from the library to the Stage. In the Property inspector, enter **pa_btn** in the Instance Name box.
- 6 Drag the `location_BK` button from the library to the Stage. In the Property inspector, enter **bk_btn** in the Instance Name box.

The Stage of your application should look something like the following:



- 7 Select Frame 61 in the `ActionScript` layer. In the `Actions` panel, enter the following code:

```

/* Frame Actions */
stop ();
fscommand2 ("SetSoftKeys", "Save", "Cancel");
setLocation ();

/* Handle soft key presses */
Key.removeListener (myListener);
var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1 ) {
        // Save location
        saveNewLocation ();
        gotoAndPlay ("home");
    }
    else if (keyCode == ExtendedKey.SOFT2) {
        // Cancel operation, go back to home screen
        gotoAndPlay ("home");
    }
};
Key.addListener (myListener);

/*
The saveNewLocation() function is called when the user presses "Save" (left soft key).
It updates the "location" shared object with the phone # associated with that location.
*/
function saveNewLocation () {
    // Determine which button (location) the user selected:
    var selectedButton = Selection.getFocus ();
    switch (selectedButton) {
        case "_level0.sf_btn" :
            // User selected San Francisco
            location_so.data.phoneNumber = "415-555-1212";
            break;
        case "_level0.sj_btn" :
            // User selected San Jose
            location_so.data.phoneNumber = "408-555-1212";
            break;
        case "_level0.bk_btn" :
            // User selected Berkeley
            location_so.data.phoneNumber = "510-555-1212";
            break;
        case "_level0.pa_btn" :
            // User selected Palo Alto
            location_so.data.phoneNumber = "650-555-1212";
            break;
    }
}

/*
The setLocation() function sets the section to
the currently selected location,
based on the phone number saved in the shared object.
*/
function setLocation () {

```

```
// Retrieve phone number stored in shared object:
var loc:String = location_so.data.phoneNumber;
// Extract area code from phone number:
var areaCode:String = loc.substring (0, 3);
// Based on area code, set selection focus to corresponding button/menu item:
switch (areaCode) {
    case "415" :
        Selection.setFocus (_level0.sf_btn);
        break;
    case "408" :
        Selection.setFocus (_level0.sj_btn);
        break;
    case "510" :
        Selection.setFocus (_level0.bk_btn);
        break;
    case "650" :
        Selection.setFocus (_level0.pa_btn);
        break;
}
```

8 Choose File > Save.

9 Choose Control > Test Movie to test the application in Device Central.

Open the Output window in Device Central to view feedback about which location was dialed.

Chapter 4: Creating interactivity and navigation

To interact with your Adobe Flash Lite application, a user must be able to determine which object on the screen currently has focus, navigate among objects, and initiate an action by selecting an object or another key. While these basic principles are the same as for desktop applications, some of the functionality varies for mobile devices.

User interaction and supported keys

About user interaction in Flash Lite

Flash Lite supports navigation and user interaction through the device's keypad, or through a stylus or touchscreen interface on devices that provide one. The options available to your application vary depending on the target device and content type. For more information about content types, see [“About Flash Lite content types”](#) on page 16.

The simplest way to add key-based interactivity to a Flash Lite application is default navigation, which uses the device's four-way keypad like the arrow keys or the Tab and Shift+Tab keys in a desktop application. The user moves the focus to the desired object and then presses the select key. The application includes event handler code to respond to these button events. Default navigation in Flash Lite works with buttons, input text fields, and, optionally, movie clips; it is typically best for simple user interactions such as menus. For more information about default navigation, see [“Using default navigation in Flash Lite”](#) on page 35.

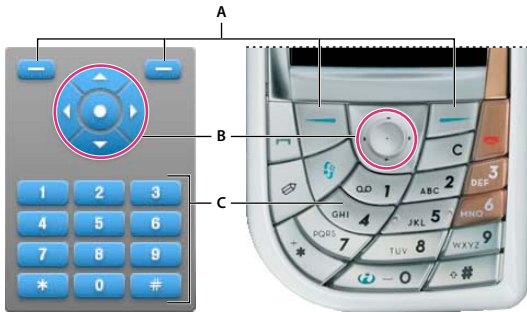
Applications can also respond to arbitrary keypress events that Flash Lite generates when a user presses a particular key. Event-based navigation allows you to create Flash Lite applications like games that have a complex user interaction model. For more information about events, see [“About key and button events”](#) on page 40.

Keys supported by Flash Lite

In addition to the alphanumeric keys available on standard telephones, most mobile devices feature a navigation keypad, which let users navigate and select items on the device screen, and two (or more) soft keys. A device's *soft keys* are multifunctional keys that use the screen to identify their purpose at any moment.

A typical navigation keypad has four navigation keys (Up, Down, Left, and Right) and a select key (typically located at the center of the keypad). Different applications can use these keys in different ways. For example, a game might use the navigation keys to let the user move a character on the screen, and then use the select key to perform another action, such as make the character jump.

The following images show the most common keys on a generic keypad and on an actual device:



A. Left and Right soft keys **B.** Navigation keypad **C.** Numeric, *, and # keys

Not all devices and Flash Lite content types support all these keys. For example, devices that support two-way navigation don't support the left and right navigation keys (see [“Default navigation modes”](#) on page 35). Also, not all devices have access to the device's soft keys.

Flash Lite supports the following keys on mobile devices:

Description	Keys	Availability
Numeric, *, #	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #	All devices
Five-way keypad	Select, up, and down	All devices
	Left and right	Devices that support four-way navigation only (see “Default navigation modes” on page 35)
Soft keys	Left and right	Devices that support the <code>SetSoftKeys</code> command
	SOFT3 - SOFT12 keys	Devices that have more than two soft keys
Keyboard keys	!, ", #, \$, %, &, ', (,), *, +, ,, -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ^, _ ' a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, {, , }, ~, Backspace	Devices that have a QWERTY keyboard

The following `System.Capabilities` properties allow you to determine the navigation and selection options available on a device:

- `hasMappableSoftKeys`
- `softKeyCount`
- `has4WayKeyAS`
- `hasQWERTYKeyboard`
- `hasStylus`
- `hasMouse`

For more information about the `System.Capabilities` class, see *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Default navigation

Using default navigation in Flash Lite

On desktop Flash applications, the Tab and Shift+Tab keys let users switch focus among objects on the screen. The arrow keys function in a similar way in some other applications. In Flash Lite, the navigation keys on the device's navigation keypad serve the same purpose as the arrow or Tab and Shift+Tab keys in a Flash desktop application. After the desired object has focus, the user can press the select key to trigger an action in the application. You define event handlers to respond when a button or movie clip is selected; for more information, see [“Handling button events”](#) on page 40.

Default navigation in Flash Lite works with buttons and input text fields. Movie clips are also included if their `tabEnabled` property is set to `true`, or if they have event handlers associated with them and their `tabEnabled` property is not set to `false`.

When an input text field has focus and the user presses the select key, Flash Lite opens the device's generic text input dialog box, in which the user can enter text.

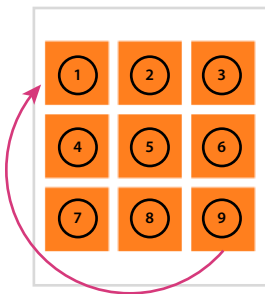
For an example of using default navigation, see [“Create a simple menu using buttons and default navigation”](#) on page 43.

Default navigation modes

Flash Lite supports three modes of default navigation: two-way, four-way, and four-way with wraparound. Different devices and Flash Lite content types support different navigation modes. For information on determining the navigation mode for a specific device and content type, see [“Playing a device video in the emulator”](#) on page 107.

Two-way navigation in Flash Lite is analogous to tab navigation in Flash desktop applications, where the Tab and Shift+Tab keys navigate among objects on the screen. The down navigation key on the device corresponds to the Tab key, and the up navigation key corresponds to the Shift+Tab key.

The default tab order in two-way navigation is generally left-to-right and top-to-bottom. For example, the following image shows a three-by-three grid of Button objects in a Flash Lite application. The numbers above each button indicate the order in which each button's will get keypad focus as the user presses the device's down navigation key repeatedly. After the button in the bottom-right corner has received keypad focus, the focus “wraps around” to the top-left button the next time the user presses the down navigation key.



Example tab order in two-way navigation

You can customize the tab order in two-way navigation using the `tabIndex` property of the Button, MovieClip, and TextField objects. For more information, see [“Controlling tab order in two-way navigation”](#) on page 40.

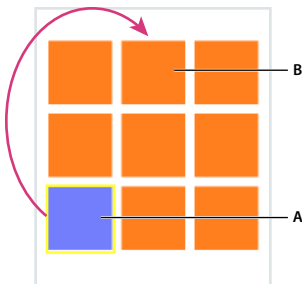
For an example of two-way navigation, see the Flash Lite Samples and Tutorials page at www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the .zip file for your version of ActionScript, download and decompress the .zip file, and then navigate to the Samples folder to access the sample file named 2-way.fla.

Four-way navigation in Flash Lite is similar to using the arrow keys on a desktop computer's keyboard to navigate among objects on the screen. The device's up, down, left, and right navigation keys correspond to the four arrow keys on a computer's keyboard. Pressing a navigation key moves the keypad focus to the object located in that direction, if one exists. If no object exists in that direction, then the keypad focus does not change from the current object.

Note: The `tabIndex` property is not supported on devices that support four-way navigation, but `tabEnabled` and `tabChildren` are, which is different from how these properties work in Flash desktop applications.

For an example that uses four-way navigation, see the Flash Lite Samples and Tutorials page at www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the .zip file for your version of ActionScript, download and decompress the .zip file, and then navigate to the Samples folder to access the sample file named 4-way.fla.

Four-way navigation with wraparound functions like a combination of standard four-way navigation and two-way navigation. Like standard four-way navigation described previously, users change keypad focus using the device's four-way navigation keys. The difference is that, similar to two-way navigation, keypad focus "wraps around" to the object on the opposite side of the screen. For example, in the image below, the button with the current keypad focus is located at the bottom-left corner of the screen. If the user presses the down navigation key, the next button to receive focus is located in the middle of the top row of buttons.



A. Button with current focus B. Next button to receive focus after user presses navigation key

You can test the behavior of two-way and four-way navigation modes in the Adobe Device Central emulator using the samples named 2-way.fla and 4-way.fla located at www.adobe.com/go/learn_flt_samples_and_tutorials. On the Samples and Tutorials page, locate, download and decompress the .zip file for your Flash Lite version, and then navigate to the Samples folder to access the samples. Each sample file consists of the same three-by-three grid of buttons, as discussed previously. The only difference between the sample files is that each FLA file is configured to target a combination of device and Flash Lite content type that supports the navigation mode (two-way or four-way).

To use each sample file, open it in Flash and test it in the Adobe Device Central emulator (select Control > Test Movie). Click the arrow keys on the emulator's keypad (or press the arrow keys on your keyboard) to see how each navigation mode affects user navigation.



Array of buttons in sample file

About the focus rectangle

By default, Flash Lite draws a yellow rectangle around the button or input text field that has focus. Movie clips are also included if their `tabEnabled` property is set to `true`, or if they have event handlers associated with them and their `tabEnabled` property is not set to `false`.

The focus rectangle lets the user know which object on the screen will respond when the user presses the device's select key. For example, the following image shows the focus rectangle drawn around a button that has the current keypad focus:



Button with current focus and default focus rectangle

For buttons and movie clips, the focus rectangle's bounding box is determined by the object's *hit area*—the invisible region that (in Flash desktop applications) defines the part of the button or movie clip that responds to mouse clicks. For input text fields, the focus rectangle's bounding box is determined by the text field's dimensions.

You can customize the color of the focus rectangle or disable it. For more information, see “[Customizing the focus rectangle](#)” on page 38.

Guidelines for using default navigation

The following are guidelines and considerations for using default navigation in your Flash Lite applications.

- If you disable the default focus rectangle by setting `_focusRect` to `false`, be sure to provide an alternative focus indicator for your buttons, input text fields, and tab-enabled movie clips. For buttons, you can do this by adding a visual element to the button's “over” state—the part of a button object's timeline that's displayed when the button has focus. For an example of this technique, see the sample application in “[Create a simple menu using buttons and default navigation](#)” on page 43. For input text fields, you can use the Selection object to determine when the text field has received focus and display the desired focus indicator. For an example, see the sample application discussed in “[Controlling focus with ActionScript](#)” on page 38.

- Have at least two objects (input text fields, buttons, tab-enabled movie clips, or a combination) on the screen at the same time. If the screen contains only one input text field, button, or tab-enabled movie clip, the user can't change the focus and may feel stuck in the user interface.

If a screen in your application contains only a single button for user interaction, consider detecting a keypress event rather than using button events. For more information, see [“About key and button events”](#) on page 40.

- When appropriate, consider using the `Selection.setFocus()` method to set the initial focus to a specific object on the screen. This can help guide the user through the interface and reduce the amount of key navigation they must perform. For example, suppose that a screen in your application contains an input text field. Normally, for the user to enter a value in the text field, they would first press a navigation key to give the text field focus, and then press the select key to open the text input dialog box. You could use the following ActionScript to automatically set the keypad focus to the input text field:

```
Selection.setFocus(inputTxt);
```

For more information about controlling focus with ActionScript, see [“Controlling focus with ActionScript”](#) on page 38.

- The alignment of objects on the screen is important with default navigation. When objects are offset from each other on the screen, the sequence in which they get focus may not be intuitive for your users. You can prevent this by making objects consistent in size and aligning them vertically and horizontally as much as possible. With two-way navigation, you can also control the sequence using the `tabIndex` property; for more information, see [“Controlling tab order in two-way navigation”](#) on page 40.

Customizing the focus rectangle

The focus rectangle is a default yellow highlight that indicates which button or input text box is currently selected. Movie clips are also included if their `tabEnabled` property is set to `true`, or if they have event handlers associated with them and their `tabEnabled` property is not set to `false`. For more information, see [“About the focus rectangle”](#) on page 37.

You can disable the default focus rectangle behavior by setting the global `_focusRect` property to `false`. You can also disable the focus rectangle for specific buttons or movie clips (see `focusRect` (`Button._focusRect` property) and `focusRect` (`MovieClip._focusRect` property)) in the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

You can also change the color of the focus rectangle from the default yellow to any other color. To do this you use the `SetFocusRectColor` command, which takes RGB values as parameters. For example, the following code changes the color of the focus rectangle to red:

```
fscommand2("SetFocusRectColor", 255, 0, 0);
```

Controlling focus with ActionScript

You can use the `Selection` ActionScript object to get and set the current keypad focus, or to be notified when an object receives or loses keypad focus. This is useful, for example, if you want to automatically set the focus to a specific button when your application first loads. Or you may want to be notified when a specific object on the screen has received (or lost) keypad focus so that you can update the screen accordingly.

For example, the following code uses the `Selection.setFocus()` method to set focus to the button instance named `login_btn`:

```
Selection.setFocus(login_btn);
```

The `Selection.onSetFocus` event listener lets you determine when the keypad focus has changed. You can use this event listener, for example, to create a custom focus manager for input text fields, rather than use the default focus rectangle. The following procedure shows how to create a custom focus manager that changes the border color of the `TextField` object with focus. For a sample of the completed application (`custom_focus_manager fla`), see the Flash Lite Samples and Tutorials page at www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the .zip file for your version of ActionScript, download and decompress the .zip file, and then navigate to the Samples folder to access the sample.

Create a custom text input focus manager

- 1 In Flash, create a new mobile document and save it as `custom_focus_manager fla`.
- 2 Using the Text tool, create a text field on the Stage.
- 3 With the text field still selected, in the Property inspector, select Input Text from the Text Type pop-up menu, type **inputTxt_1** in the Instance Name text box, and select the Show Border Around Text option.
- 4 In the same manner, create another input text field below the first one with the instance name of **inputTxt_2** and select the Show Border Around Text option for the second text field.
- 5 In the Timeline, select Frame 1 in the layer named ActionScript.
- 6 Open the Actions panel (Window > Actions) and enter (or copy and paste) the following code:

```
// Disable focus rect globally:
_focusrect = false;
// Create Selection listener object:
var focusListener:Object = new Object ();
// Define onSetFocus method:
focusListener.onSetFocus = function (oldFocus, newFocus) {
    // Enable/disable selection indicator:
    if (newFocus instanceof TextField) {
        // Set border color of text field with new focus to red:
        newFocus.borderColor = 0xFF0000;
    }
    if (oldFocus != undefined && oldFocus instanceof TextField) {
        // Set border color of text field with old focus to black:
        oldFocus.borderColor = 0x000000;
    }
};
// Add listener to Selection object:
Selection.addListener (focusListener);
// Set initial focus when application loads:
Selection.setFocus (inputTxt_1);
// Enable full-screen mode:
fscommand2 ("FullScreen", true);
```

- 7 Save your changes and test the application in the emulator (Control > Test Movie).
- 8 Press the emulator's down and up arrow keys to switch keypad focus between the two text fields. The text field with focus should have a red border, and the text field without focus should have a black border. Press the select key when a text field has focus to make the text input dialog box appear.

Controlling tab order in two-way navigation

Two-way navigation in Flash Lite is analogous to tab navigation in Flash, and therefore supports the `tabIndex` property that allows you to specifically set the tab order of buttons, movie clips, and input text fields. On devices that support four-way navigation, the `tabIndex` property is not supported, so it's not possible to set tab order using the `tabIndex` property for four-way navigation.

To control tab order in two-way navigation, you assign each object's `tabIndex` property a number that specifies the object's order in the default navigation. For example, suppose that an application contains a button (`my_button`), a movie clip (`my_movieclip`), and an input text field (`my_inputTxt`). The following code establishes the tab order so that the button gets focus first, then the movie clip, and finally the input text field.

```
my_button.tabIndex = 1;  
my_movieclip.tabEnabled = true;  
my_movieclip.tabIndex = 2;  
my_inputTxt.tabIndex = 3;
```

Key and button events

About key and button events

Event handlers and event listeners specify how the application will respond to user- and system-generated occurrences. For example, when a button has focus and the user presses the select key, an `onPress` event is generated. In addition to using default navigation and responding to related events, a Flash Lite application can listen for and respond to keypress events.

Not all devices and content types support all device keys. For example, on a device that supports two-way navigation (see “[Default navigation modes](#)” on page 35) Flash Lite doesn't generate keypress events for the left and right arrow keys. For a list of keys and description of their availability, see “[Keys supported by Flash Lite](#)” on page 33.

Handling button events

You can use buttons to quickly add interactivity to your Flash Lite applications. Flash Lite supports the same button events as Flash Player on desktop computers, although some events (for example, `onDragOut`) are only available on devices that have a mouse or stylus interface. On devices that have a keypad interface only, a button must have keypad focus before it will generate any events.

Flash Lite supports the following ActionScript button events:

Button event	Description
<code>onDragOut</code>	Supported only on devices that have a mouse or stylus. Invoked when the user presses the mouse button over the button and the pointer is then dragged outside the button.
<code>onDragOver</code>	Supported only on devices that have a mouse or stylus. Invoked when the user presses and drags the mouse button outside and then over the button.
<code>onKeyDown</code>	Invoked when the button has focus and a key is pressed.
<code>onKeyUp</code>	Invoked when the button has focus and a key is released.
<code>onKillFocus</code>	Invoked when focus is removed from a button.

Button event	Description
onPress	Invoked when the user presses the select key on the device when the button has focus. Also, invoked when the user presses the mouse button over the button when the button has focus.
onRelease	Invoked when the user releases the select key on the device when the button has focus. Also, invoked when the user releases the mouse button over the button when the button has focus.
onReleaseOutside	Invoked when the mouse button is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.
onRollOut	Invoked when a button loses focus.
onRollOver	Invoked when a button receives focus.
onSetFocus	Invoked when a button receives the input focus.

The following procedure demonstrates how to create a simple application that handles button events. For an example of using buttons to create a menu, see [“Create a simple menu using buttons and default navigation”](#) on page 43.

Create a button event handler

- 1 In Flash, create a new mobile document and save it as `custom_focus_manager fla`.
- 2 Select Window > Common Libraries > Buttons to open an external library of prebuilt button symbols.
- 3 In the Library panel, double-click the classic buttons folder to open it, and then open the Circle Buttons folder.
- 4 Drag an instance of the Menu button symbol to the Stage.
- 5 In the Property inspector, in the Instance Name text box, type **btn_1**.
- 6 Drag another instance of the same button to the Stage and position it directly below the first button.
- 7 In the Property inspector, in the Instance Name text box, type **btn_2**.
- 8 In the Timeline, select Frame 1 in the layer named ActionScript.
- 9 Open the Actions panel (Window > Actions) and enter the following code:

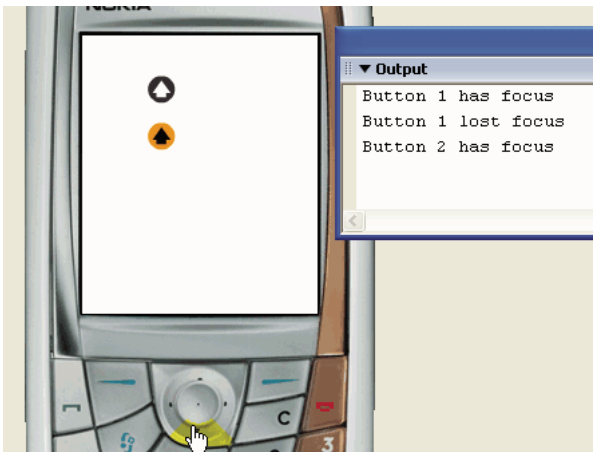
```
// Disable the focus rectangle because buttons have an over state
_focusRect = false;

// Event handlers for btn_1
btn_1.onPress = function() {
    trace("You pressed Button 1");
}
btn_1.onRelease = function() {
    trace("You released Button 1");
}
btn_1.onRollOver = function() {
    trace("Button 1 has focus");
}
btn_1.onRollOut = function() {
    trace("Button 1 lost focus");
}

// Event handlers for btn_2
btn_2.onPress = function() {
    trace("You pressed Button 2");
}
btn_2.onRelease = function() {
    trace("You released Button 2");
}
btn_2.onRollOver = function() {
    trace("Button 2 has focus");
}
btn_2.onRollOut = function() {
    trace("Button 2 lost focus");
}
}
```

10 Test the application in the emulator (Control > Test Movie).

Watch the messages in the Output panel as you press the up and down arrow keys on the emulator's keypad.



Other types of objects support different events; for example, the `TextField` object includes an `onChanged` event that is invoked when the content of a text field changes. You can write event handler code for these events using the same format as the button event handlers in this procedure. For more information about the events supported for text fields and movie clips, see the `TextField` and `MovieClip` entries in the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Create a simple menu using buttons and default navigation

To create a simple menu using buttons and default navigation, you need to design the menu and then write event handling code for it. To create the menu, you'll use three button symbols, one for each menu option. Then you'll write event handling code that displays a message when the user rolls over each menu item—that is, when the user gives focus to the corresponding button—and when the user selects the menu item by pressing the select key on the device. For more information about handling button events in Flash Lite, see “[Handling button events](#)” on page 40.

Start with a partially completed Flash document. You can change these settings to target a different device and content type (see “[Using the emulator](#)” on page 103).

- 1 Download and open the file named `simple_menu_start.fla` located at www.adobe.com/go/learn_flt_samples_and_tutorials. On the Samples and Tutorials page, locate, download and decompress the .zip file for your Flash Lite version, and then navigate to the Samples folder to access the sample.
- 2 Open the Library panel (Window > Library).
Notice that the Library contains three button symbols named News Button, Weather Button, and Sports Button.
- 3 In the Timeline (Window > Timeline), select the layer named Menu Buttons.
- 4 Drag an instance of the News Button symbol from the Library panel to the Stage.
- 5 In the Property inspector, in the Instance Name text box, type **btn_news**.
- 6 Repeat steps 4 and 5 for the Sports and Weather buttons, naming them **btn_sports** and **btn_weather**, respectively.
- 7 Align the three buttons vertically, as shown in the following example:



- 8 In the Tools panel, select the Text tool and create a text field along the bottom of the Stage.
This text field displays a short message when the user rolls over each menu item.
- 9 With the new text field still selected, do the following in the Property inspector:
 - a Select Dynamic Text from the Text Type pop-up menu.
 - b Type **txt_status** in the Instance Name text box.

The Stage should look something like the following image:



10 In the Timeline, select Frame 1 in the layer named `ActionScript`.

11 Open the Actions panel (Window > Actions) and enter the following code:

```
// Disable the focus rectangle because buttons have an over state
_focusRect = false;

btn_news.onRollOver = function() {
    txt_status.text = "Press to select News";
}
btn_news.onPress = function() {
    txt_status.text = "You selected News";
}
btn_sports.onRollOver = function() {
    txt_status.text = "Press to select Sports";
}
btn_sports.onPress = function() {
    txt_status.text = "You selected Sports";
}
btn_weather.onRollOver = function() {
    txt_status.text = "Press to select Weather";
}
btn_weather.onPress = function() {
    txt_status.text = "You selected Weather";
}
```

12 Select Control > Test Movie to preview the application in the emulator.

Click the down arrow key on the emulator with your mouse (or press the down arrow key on your computer's keyboard) to navigate between menu options; to select a menu item, click the emulator's select key by using your mouse (or press the Enter key on your computer's keyboard).



Handling key events

Flash Lite generates keypress events in response to the user pressing keys on the device. You can write key handler code to respond to these events. For a list of the keys that Flash Lite supports, see “[Keys supported by Flash Lite](#)” on page 33.

The following table lists commonly used device keys and the corresponding ActionScript key codes and key code constants for those keys:

Device key	ActionScript key code/key code constant
Select key	Key.ENTER
Up navigation key	Key.UP
Down navigation key	Key.DOWN
Left navigation key	Key.LEFT
Right navigation key	Key.RIGHT
Left soft key	ExtendedKey.SOFT1 (or soft1)
Right soft key	ExtendedKey.SOFT2 (or soft2)
0	48
1	49
2	50

Device key	ActionScript key code/key code constant
3	51
4	52
5	53
6	54
7	55
8	56
9	57
*	56
#	51

Writing an event listener

Event listeners let an object, called a *listener object*, receive events broadcast by another object, called a *broadcaster object*. The broadcaster object registers the listener object to receive events generated by the broadcaster. For more information, see “Using event listeners” in *Learning ActionScript 2.x in Flash*.

An easy way to handle keypress events is to create a key listener object that defines an `onKeyDown` or `onKeyUp` function, and then register that object with the `Key.addListener()` method. The following example code defines a key listener that responds when the user presses the right navigation key on the device:

```
var myListener:Object = new Object();
myListener.onKeyDown = function() {
    if (Key.getCode() == Key.RIGHT) {
        trace("You pressed the right arrow key");
    }
}
Key.addListener(myListener);
```

Handle keypress events through a key listener

- 1 In Flash, create a new mobile document and save it as `keylistener fla`.
- 2 Select the layer in the Timeline named `Content`.
- 3 Using the Oval tool, create an oval or circle on the Stage and convert it to a movie clip.
- 4 With the new movie clip selected, in the Property inspector, type **circle** in the Instance Name text box.
- 5 In the Timeline, select the first frame in Layer 1.
- 6 Open the Actions panel (Window > Actions), and enter the following code:

```

var myListener:Object = new Object();
myListener.onKeyDown = function() {
    if (Key.getCode() == Key.LEFT) {
        circle._x -= 10;
    } else if (Key.getCode() == Key.RIGHT) {
        circle._x += 10;
    } else if (Key.getCode() == Key.UP) {
        circle._y -= 10;
    } else if (Key.getCode() == Key.DOWN) {
        circle._y += 10;
    }
};
Key.addListener(myListener);

```

7 Test the application by selecting Control > Test Movie.

Press the four navigation keys on the emulator's keypad (or the corresponding arrow keys on your keyboard) to make the circle move around the Stage.

Using the soft keys

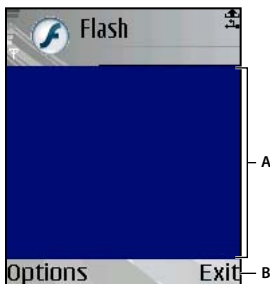
To use the soft keys in your Flash Lite application, you must first call the `SetSoftKeys` command. Then, Flash Lite generates an `ExtendedKey.SOFT1` event when the user presses the Left soft key and an `ExtendedKey.SOFT2` event when the user presses the Right soft key. You write ActionScript event handler code that responds to these events and takes the desired action.

The `SetSoftKeys` command takes two parameters that specify the labels for the Left and Right soft keys, respectively, that appear when your application is *not* running in full-screen mode. For applications running in full-screen mode, the labels that you specify are not visible, so you must create your own labels and position them on the Stage where the soft keys are located.

For example, consider the following `SetSoftKeys` command call:

```
fscommand2("SetSoftKeys", "Options", "Exit");
```

The following example shows the result of using this command in an application running on an actual device in normal (not full-screen) mode:



A. Available screen area in non-full-screen applications B. Soft key labels displayed by device

If you enable full-screen mode—that is, if you call `fscommand("fullscreen", true)`—the labels that you specify as parameters to the `SetSoftKeys` command are not visible. Consequently, in full-screen mode applications, you must create your own soft key labels, as shown in the following example:



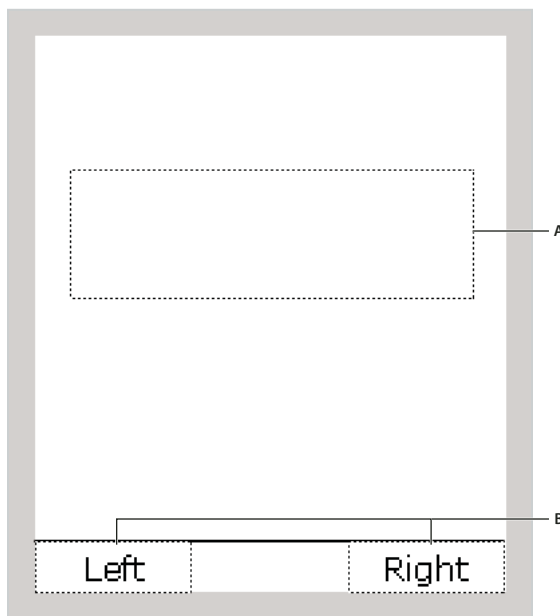
Custom soft key labels

For more information about the `SetSoftKeys` command, see the `fscommand2` function entry in the *Flash Lite 2.x ActionScript Language Reference*.

Use the soft keys in an application

- 1 In Flash, create a new mobile document and save it as `softkey.fl`.
- 2 Using the Text tool, create a static text field named `Left` and position it in the lower-left corner of the Stage, above the Left soft key on the device.
- 3 Create another static text field named `Right`, and position it in the lower-right corner of the Stage, above the Right soft key on the device.
- 4 Using the Text tool, create a dynamic text field, and position it in the middle of the Stage.

Your document's Stage should look like the following example:



A. Dynamic text field B. Soft key labels

- 5 With the dynamic text field still selected, in the Property inspector, type **status** in the Instance Name text box.
- 6 Open the Actions panel (Window > Actions) and, in the Timeline, select Frame 1 in Layer 1.
- 7 In the Actions panel, type the following code:

```
fscommand2("SetSoftKeys", "Left", "Right");  
fscommand2("FullScreen", true);
```

- 8 Create and register an object to respond to keypress events (see [“Handle keypress events through a key listener”](#) on page 46) by entering the following code in the Actions panel:

```
var myListener:Object = new Object();  
myListener.onKeyDown = function() {  
    if (Key.getCode() == ExtendedKey.SOFT1) {  
        // Handle left soft keypress event.  
        status.text = "You pressed the Left soft key.";  
  
    } else if (Key.getCode() == ExtendedKey.SOFT2) {  
        // Handle right soft keypress event.  
        status.text = "You pressed the Right soft key.";  
    }  
};  
Key.addListener(myListener);
```

- 9 Select Control > Test Movie to test the application in the emulator.

To test the application, click the Left and Right soft keys on the emulator with your mouse, or press the Page Up and Page Down keys on your keyboard.



Chapter 5: Working with text and fonts

You can add static and dynamic text fields and add input text fields to your Macromedia Flash Lite 2.x from Adobe and Adobe Flash Lite 3.x applications.

Text

About text in Flash Lite

Flash Lite 2.x and 3.x support the following text features:

- Static, dynamic, and input text fields

At run time, the contents of a static text field can't change, but the contents of a dynamic or input text field can change. Input text fields allow users to enter text. Flash Lite 2.x and 3.x support inline text input on most devices. On devices that support complex languages and in Flash Lite 2.0, input text fields use the device's generic text input mechanism. For more information about input text fields, see "[Using input text fields](#)" on page 51.
- Embedded and device fonts

You can have Flash Lite render text fields using font outlines that you embed in the SWF file or using fonts available on the device. For more information about font rendering methods, see "[Font rendering methods in Flash Lite](#)" on page 58.
- Unicode text encoding

Flash Lite can display text in any language as long as fonts containing the required character glyphs are available. For information on multilanguage authoring in Flash, see "Creating Multilanguage Text" in *Using Flash*.
- Partial support for HTML formatting and the TextFormat class properties
- Scrolling text

Flash Lite does not support all the text features that are in the desktop version of Flash Player. Flash Lite has the following limitations:

- Advanced anti-aliasing, the enhanced font rendering technology available in Macromedia Flash Player 8 from Adobe and later, is not supported.
- Text formatting is supported, but only the color, face, size, bold, and italic options are available. Also, formatting is not displayed if the device text font does not include the selected option. For example, a field formatted as italic appears as regular text when the device font does not include an italic version.
- Device text cannot be masked, used as a mask, or rendered with transparency.
- The Render Text As HTML formatting option is partially supported for input and dynamic text fields. Text is displayed without visible HTML tags, but formatting is honored only for the following tags: `p`, `br`, `sbr`, `font` (with `face`, `color`, and `size` attributes), `b`, and `i`.
- Flash Lite does not support Cascading Style Sheets (CSS).
- Flash components, including `Label`, `TextArea`, and `TextInput`, are not supported.
- The `TextField` and `TextFormat` objects are partially supported, and additional limitations apply for Arabic, Hebrew, and Thai. For more information, see the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

- The XML and XMLNode objects are supported.

Flash Lite 2.1 and later adds support for inline text input, predictive text engines, and XMLSocket:

- Inline text input support lets the user enter text directly into text fields.
- Predictive text support enables functionality such as word completion and candidate lists. Flash Lite 2.1 and later supports the top predictive text engines (such as T9, eZiTap/eZiText, and iTap) on any platform, as long as they are implemented similarly to the standard APIs provided by the predictive text engine vendors.
- XMLSocket support extends the Flash desktop support to Flash Lite, and enables developers to create continuous, low-latency data connections for applications such as games and chat.

Creating and formatting text

You create and format text in Flash Lite as you would for a Flash desktop application.

For more information about working with text in Flash, see the following topics in Using Flash:

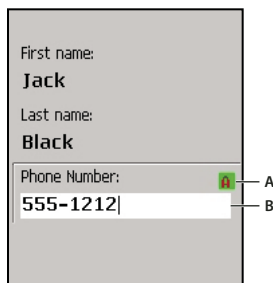
- “Creating text”
- “Setting text attributes”
- “Editing text”
- “Controlling text with ActionScript”

For a list of text features that are not supported in Flash Lite, see “[About text in Flash Lite](#)” on page 50.”

Text input

Using input text fields

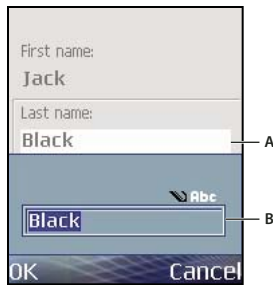
Flash Lite 2.1 and later supports inline text input. This feature lets users edit text fields directly in the Flash Lite application, rather than in a separate text input box as in earlier versions. For example, the following figure shows how an inline input text field appears on a Series 60 device from Nokia:



A. Input mode indicator B. Text field being edited inline

Flash Lite 2.1 and later supports inline text input for Latin and Asian languages but not complex languages, which include right-to-left languages such as Arabic, Hebrew, Urdu, Farsi, and Yiddish, as well as some Asian languages.

On devices running Flash Lite 2.0 and Flash Lite 1.1, and in the Adobe Device Central emulator for all versions, users edit the contents of input text fields using a modal dialog box that appears over the Flash Lite content. For example, the following figure shows an example of a text input dialog box on a Symbian™ Series 60 device running Flash Lite 2.0:



A. Text input dialog box B. Text field being edited

In general, existing Flash Lite 2.0 and Flash Lite 1.1 applications can work without modification in Flash Lite 2.1 and later. In place of the modal dialog box, users can simply edit input text fields inline. However, legacy content should be re-authored to use the features that are available only in Flash Lite 2.1 and later, such as the ability to set text selections or the `activateTextField` command.

For more information about the input text dialog box, see [“About the device’s input text dialog box \(Flash Lite 2.0\)”](#) on page 55.”

Using inline text input (Flash Lite 2.1 and later)

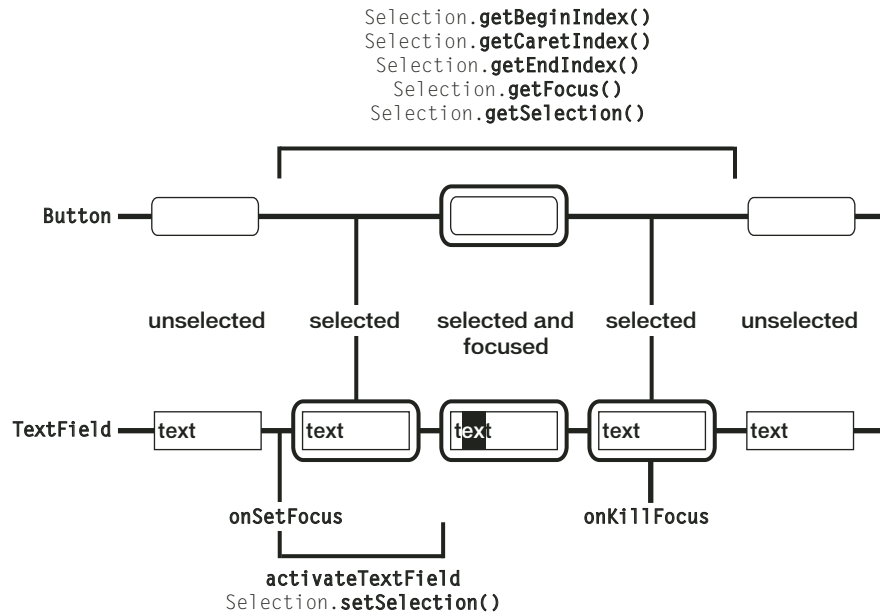
Flash Lite 2.1 and later supports inline text input, which allows users to enter and edit text directly in an input text field. While entering text in a Flash Lite application, users interact with the device’s native input method editor (IME)—the Flash Lite player does not process the user input itself.

Note: The Flash Lite emulator on Adobe Device Central does not show inline text. For more information, see [“Testing inline text in Flash Lite 2.1 and later”](#) on page 102.

While an input text field is active, the Flash Lite 2.1 and 3.x player runs in *restricted* mode. In this mode, the device’s native IME handles all keypress events, and no ActionScript is processed. In addition, all animation, sound, and stop-video-playing events are ignored. After a text field is deactivated, Flash Lite resumes normal operating mode.

Before a text field can accept text input from a user, it must be *activated*. A user can activate an input text field by giving it selection focus and then either pressing the device’s select key or pressing a soft key for which the application sets the text entry to the focused state. (For details about activating a text field, see [“Activating input text fields with ActionScript”](#) on page 53.)

The following figure shows the different states of an input text field: not selected, selected (but not activated), and activated:



Inline input text fields and navigation

An activated text field contains a pointer that indicates the current insertion point. The user can change the pointer's location within the text field using the device's navigation keys.

By default, Flash Lite draws a focus rectangle around the text field that has the current focus. This focus rectangle can sometimes obscure an activated text field's flashing pointer or insertion indicator. For this reason, it's recommended that you disable the focus rectangle (by setting `_focusRect` to `false`) and use a custom focus indicator. For an example of an application that uses a custom focus indicator, see the Flash Lite Samples and Tutorials page at www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the ZIP file for your version of ActionScript, download and decompress the ZIP file, and then navigate to the Samples folder to access the sample.

Activating input text fields with ActionScript

A user can activate an input text field that has keypad focus by pressing the device's select key. A Flash Lite application can also automatically activate an input text field when it receives focus using the `activateTextField` command. This command activates the currently selected text field; if there is no selected field when the command is executed, nothing happens.

The most common place to call `activateTextField` is from within a `Selection.onSetFocus` handler or a `TextField.onSetFocus` handler. For example, suppose that your application contains two (or more) input text fields on the Stage. The following code automatically activates the text field that receives focus:

```
var focusListener:Object = new Object ();
focusListener.onSetFocus = function (oldFocus, newFocus) {
    // Call activateTextField:
    fscommand ("activateTextField", "");
};
TextField1.addListener (focusListener);
```

You can also use the `TextField.prototype.onSetFocus` handler to activate all text fields whenever they receive focus.

It is also possible to use keys other than the device's select key to trigger the `activateTextField` command. The following code activates a text field for all number keys, which makes it easier to enter—for example, the letter “a”. For example, if the application includes `activateTextField` in the `TextField.onSetFocus` handler, the user would have to press Select and then 2; this code allows the user press 2 twice, which is more intuitive.

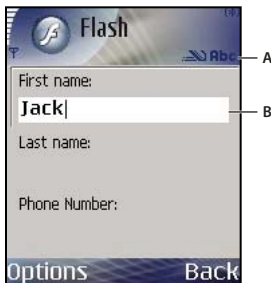
```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.getCode() >= 48 && Key.getCode() <= 57 ) {
        fscommand("activateTextField", "");
    }
}
Selection.addListener (keyListener);
};
```

For a completed sample application that uses this technique, see the Inline Text Input sample at www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the ZIP file for your version of ActionScript, download and decompress the ZIP file, and then navigate to the Samples folder to access the sample.

Input mode indicator

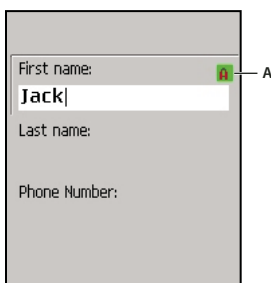
To assist users with common text input tasks, most devices provide several input modes, such as predictive or manual (*triple tap* or *multi tap*) text entry, or all number-only modes.

When Flash Lite is running in full screen mode, the device displays a letter A for alpha input mode and a number sign (#) for numeric input mode. When Flash Lite is not running in full screen mode, the device could draw an input mode indicator on a status bar or some other location on the display. For example, the following image shows the input mode indicator in non-full screen mode on the Series 60 stand-alone version of Flash Lite 2.1:



A. Input mode indicator, non-full screen mode B. Text field being edited

When the player is running in full-screen mode, the device may draw an input mode indicator to a location of its choice on the display. For example, the following figure shows the input mode indicator in full screen mode on the Series 60 stand-alone version of Flash Lite 2.1:



A. Input mode indicator, full screen mode

The input mode indicator for full screen applications shown in the previous figure is a reference implementation for the stand-alone Series 60 player. The specific indicator, if any, that appears is determined by the device.

About the device's input text dialog box (Flash Lite 2.0)

To open the device's input dialog box, users must first give an input text field focus, and then press their device's select key.

The text input dialog box is modal, which means that the user can't interact with the Flash content while the dialog box has focus. Flash Lite also pauses the playhead in the Flash application while the dialog box has focus.

If the user selects OK (the Left soft key), the text input dialog box closes, and Flash Lite automatically assigns the text to the input text field. If the user selects Cancel (the Right soft key), no text is assigned to the input text field.

The Adobe Device Central emulator mimics the features of the text input dialog box when you test your application in the Flash authoring tool. The following image shows the text input dialog box running in the emulator:

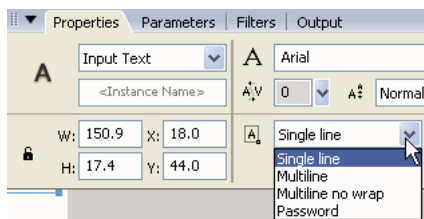


Text input dialog box running in the emulator

For an example of using an input text field in an application, see [“Create a text field sample application”](#) on page 57.

Specifying types of input text fields

Flash Lite supports single line, multiline, and password input text fields. You specify an input text field's type by using the Line Type pop-up menu in the Property inspector, as the following image shows:



The line type that you specify for an input text field determines the behavior of the device's input text dialog box when a user edits the text field.

For example, when a user edits a single line input text field, the device's input text dialog box shows a single line input text box. The input text box scrolls horizontally if the user enters more characters than can be displayed.

Restricting character input

You can use the `SetInputTextType` command to restrict the characters that the user can enter in the text input dialog box. For example, suppose an application contains an input text field for users to provide a numeric value, such as their age, and that the input text field has the variable name `ageVar`. To ensure that the user enters only numeric values in the text input dialog box, you could add the following code to your application:

```
fscommand2 ("SetInputTextType", "ageVar", "Numeric");
```

When users open the input text dialog box, they can enter only numeric values in the text fields.

For more information, see `SetInputTextType` in the `fscommand2` function entry in the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Input text dialog boxes (Flash Lite 2.0)

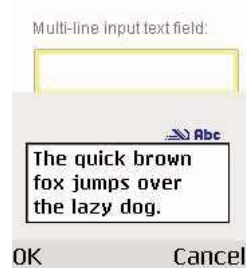
In Flash Lite 2.0, the user enters text into a separate modal dialog box (controlled by the host application, not Flash Lite) rather than interacting with the onscreen text field directly. During this interaction, the Flash Lite player is effectively paused until the user exits the dialog box.

(In Flash Lite 2.1 and later, the user can enter text directly into the onscreen text field.)

The following image shows a device's input text dialog box for a single line input text field in a Flash Lite 2.0 application:



When a user edits a multiline input text field, the device's input text dialog box expands as necessary to display all the text the user enters, as the following image shows:



When a user edits a password input text field, the device's input text dialog box displays each character that the user enters. When the user clicks OK, the entire password is masked with asterisks, as the following image shows:

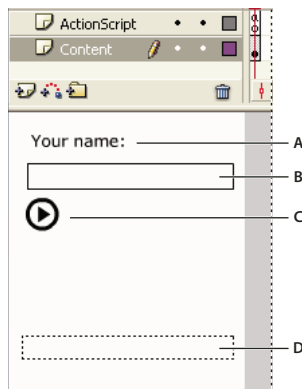


Create a text field sample application

This simple application gets text input from the user, and then formats and displays that text in an HTML-enabled dynamic text field. The application also uses the `SetFocusRectColor` command to change the focus rectangle color from the default color (yellow) to black.

For a completed sample application (textfield_example.fla) that uses this technique, see www.adobe.com/go/learn_flt_samples_and_tutorials. Locate the ZIP file for your version of ActionScript, download and decompress the ZIP file, and then navigate to the Samples folder to access the sample.

- 1 In Flash, create a new mobile document and save it as textfield.fla.
- 2 Using the Text tool in the Tools panel, create a single-line text field across the top of the Stage.
- 3 With the text field still selected, in the Property inspector, select Input Text from the Text Type pop-up menu, select Use Device Fonts from the Font Rendering Method pop-up menu, and type **inputTxt** in the Instance Name text box.
- 4 Create another text field below the first that is several times taller than the first one, as shown below:



- 5 With the second text field selected, in the Property inspector, select Dynamic Text from the Text Type pop-up menu, select Multiline from the Line Type pop-up menu, select the Render Text As HTML option, select Use Device Fonts from the Font Rendering Method pop-up menu, and type **messageTxt** in the Instance Name text box.
- 6 In the Timeline, select Frame 1 on Layer 1.
- 7 Open the Actions panel (Window > Actions) and enter the following code:

```
Selection.setFocus(inputTxt);
fscommand2("SetFocusRectColor", 0, 0, 0);
inputTxt.onChanged = function() {
    messageTxt.htmlText = "You entered: <i>" + this.text + "</i>";
}
```

The `Selection.setFocus()` method sets the initial focus to the input text field (`inputTxt`). Next, the `fscommand2()` function call specifies a custom focus rectangle color. Lastly, the input text field's `onChanged` event handler, called whenever the contents of the input text field changes, formats and displays the text that the user entered in the `messageTxt` text field.

- 8 Save your changes and start the application in the emulator (Control > Test Movie).
- 9 To use the application, press the emulator's select key to open the text input dialog box and enter some text using your computer's keyboard. Then click OK to close the dialog box. The text that you entered appears in the `messageTxt` text field in italics.

Font rendering

Font rendering methods in Flash Lite

Flash Lite can render text field fonts in any of the following ways:

Use fonts that are available on the device You can apply a font to a text field that you know is available on the device, or you specify one of the three generic device fonts (`_sans`, `_serif`, or `_typewriter`) that are available in the Font pop-up menu. If you select a generic device font, Flash Lite tries to match the selected generic font to a font on the device at run time (for example, `_sans` is mapped to a sans serif font, if available).

Render the font as a bitmap Flash Lite renders bitmap text by aligning font outlines to pixel boundaries, which makes text readable at small point sizes (such as 10 points or smaller). This option requires that you include font outlines in the published SWF file for the selected font. (See [“Embedding font outlines in SWF files”](#) on page 60.)

Render the font as anti-aliased vectors Flash Lite renders anti-aliased text using vector-based representations of font outlines, which you embed in the published SWF file. (See [“Embedding font outlines in SWF files”](#) on page 60.)

You select a font rendering method for a text field using the Font Rendering Method pop-up menu located in the Property inspector. The Font Rendering Method pop-up menu contains five rendering options; however, only three of those are available to Flash Lite developers. The other two methods (Anti-Alias For Readability and Custom Anti-Alias) are available only to applications that are targeting Flash Player 8 or later on desktop computers.

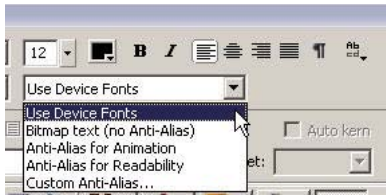
Below are some guidelines to consider when you choose between anti-aliased, bitmap, and device text:

- If you're using embedded fonts with dynamic or input text fields, embed the outlines only for the characters that you need to display. This will help to reduce file size. For example, if you're using an input text field to capture a user's age (a number), include only the font outline for number characters (0-9). In this case, consider restricting the character input to numbers.
- The Adobe Device Central emulator does not emulate device fonts, unless you have the same fonts installed on the computer that you're using to develop the content. Therefore, the layout and appearance of your device text field might be different in the emulator than on the device.
- If you apply one of the generic device font faces (`_sans`, `_serif`, or `_typewriter`), Flash Lite tries to find a similar font on the device to display the text. However, because mobile devices typically have fewer fonts and font styles than a desktop computer, a font such as `_sans` might not map to a sans serif font. You must test the application on each target device to determine the proper font settings.

Anti-aliased text in Flash Lite is, essentially, a complex vector shape. Like any vector shape, it takes some processing power to render. Because processing speed on most devices is relatively slow, animating a lot of anti-aliased text may degrade application performance. To improve performance, try temporarily lowering the Flash Lite player's rendering quality during the animation, and then returning it to the higher rendering quality when the animation is complete.

Select a font rendering method for a text field

- 1 Select a text field on the Stage.
- 2 In the Property inspector, select one of the following options from the Font Rendering Method pop-up menu:



- Select Use Device Fonts to have Flash Lite use a font that is available on the device. No font data is embedded in the published SWF file.
- Select Bitmap Text (No Anti-Alias) to have Flash Lite align font outlines along pixel boundaries, which makes small text appear crisp and clear. This option requires that Flash embed font outlines in the published SWF file. (See “[Embedding font outlines in SWF files](#)” on page 60.)
- Select Anti-Alias For Animation to have Flash Lite anti-alias the text field's font according to the current rendering quality setting (see “[Flash Lite rendering quality and anti-aliased text](#)” on page 59). This option requires that Flash embed font outlines in the published SWF file.

Flash Lite rendering quality and anti-aliased text

Use the `MovieClip._quality` property to control how Flash Lite renders vector graphics and bitmap graphics. Possible values are "LOW", "MEDIUM", "HIGH", and "BEST". The default value is "HIGH". You can specify this property for each movie clip object. To specify this property globally, place the code `_quality = "lowMediumHighOrBest"` on Frame 1.

Flash Lite renders anti-aliased text using vector representations of font outlines. If you want anti-aliased text to appear as smooth as possible, leave the rendering quality set to high. The following figure shows an anti-aliased text field (Arial, 24 point) rendered at high, medium, and low qualities:

*High
quality
setting*

To maximize animation performance and frame rate—for example, during an intensive animation or tween sequence—you can temporarily set the rendering quality to a lower setting and return it to the previous setting after the animation has completed.

To set the quality of input text, call the `fscommand2("SetInputTextType")` command. For more information, see the entry in the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Embedding font outlines in SWF files

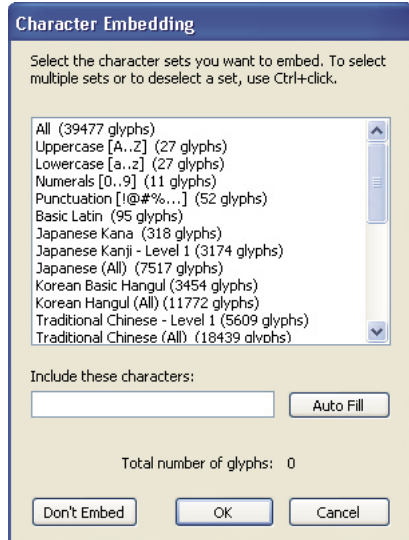
To render a text field's font, Flash Lite can either use fonts that are available on the device or use font outlines that are embedded in the published SWF file (see “[Font rendering methods in Flash Lite](#)” on page 58). Embedding font outlines in the SWF file ensures that the text field's font appears the same on all target platforms, but results in a larger file size. Flash Lite requires font outlines to render either bitmap (no anti-alias) or anti-aliased text.

For static text fields that use the anti-alias or bitmap font rendering methods, Flash automatically embeds the font outlines required to display the text field's contents. For example, if a static text field contains the word *Submit*, Flash automatically embeds the font outlines required to display those six characters (*S*, *u*, *b*, *m*, *i*, and *t*). Because the contents of a static text field can't change, the SWF file needs to include font outlines only for those characters.

For dynamic and input text fields that use the anti-alias or bitmap font rendering methods, you must specify the characters whose font outlines you want to embed in the published SWF file. The contents of those types of text fields can change during playback; consequently, Flash can't assume which font outlines need to be available. You can include font outlines for all characters in the selected font, a range of characters, or specific characters. You use the Character Embedding dialog box to specify which characters you want to embed in the published SWF file.

Embed font outlines for a dynamic or input text field

- 1 Select the dynamic or input text field on the Stage.
- 2 In the Property inspector, select Bitmap (No Anti-Alias) or Anti-Alias For Animation from the Font Rendering Method pop-up menu.
- 3 Click the Embed button located next to the Font Rendering Method menu to open the Character Embedding dialog box.



- 4 Select the characters you want to embed from the list, type the characters that you want to embed in the text box, or click Auto Fill to include the characters that are in the selected text field.
- 5 Click OK.

Scrolling text

Creating scrolling text

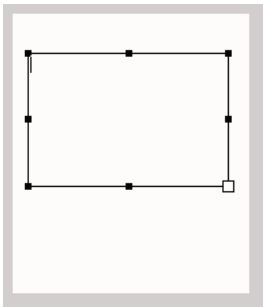
Flash Lite supports the `TextField.scroll` and `TextField.maxscroll` properties, which let you create scrolling text fields. The `scroll` property specifies the first visible line in a text block; you can get and set its value. For example, the following code scrolls the text field whose variable name is `story_text` down by five lines:

```
story_text.scroll += 5;
```

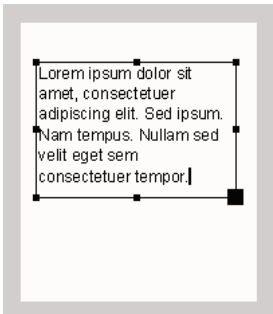
The `maxscroll` property specifies the first visible line in a text block when the last line of the text is visible in the text block; this property is read-only. You can compare a text field's `maxscroll` property to its `scroll` property to determine how far a user has scrolled within a text field. This is useful if you want to create a scroll bar that provides feedback about the user's current scroll position relative to the maximum scroll position.

Create a scrolling text field and control it with ActionScript

- 1 In Flash, create a new mobile document.
- 2 Using the Text tool, drag a text field approximately the size shown in the following image on the Stage:



- 3 Select Multiline from the Line Type pop-up menu in the Property inspector.
- 4 Select Dynamic Text from the Text Type pop-up menu in the Property inspector.
- 5 Select Use Device Fonts from the Font Rendering Method pop-up menu in the Property inspector.
- 6 Select Text > Scrollable to make the text field scrollable.
- 7 Type **story** in the Instance Name text box in the Property inspector.
- 8 Double-click inside the text field, and enter enough text so that one or more lines of text extend below its lower edge.



- 9 In the Timeline, select the first frame on Layer 1, and open the Actions panel

10 (Window > Actions). Enter the following code in the Actions panel:

```
var keyListener:Object = new Object();  
keyListener.onKeyDown = function() {  
    var keyCode = Key.getCode();  
    if (keyCode == Key.DOWN) {  
        story.scroll++;  
    }  
    else if (keyCode == Key.UP) {  
        story.scroll--;  
    }  
};  
Key.addListener(keyListener);
```

11 Select Control > Test Movie to test the application in the Adobe Device Central emulator.

Click the up and down navigation keys on the emulator (or the Up Arrow and Down Arrow keys your computer's keyboard) to scroll the text up or down.

Chapter 6: Working with sound

Adobe Flash Lite supports two types of sound: standard (*native*) Flash sound and device sound. Native sounds are played directly by the Flash Lite player, just as they are in the desktop version of Adobe Flash Player. Device sounds, in contrast, are played directly by the device, rather than by the Flash Lite player. Some examples of device sound formats include MIDI or MFi, but supported sound formats vary from device to device.

Device sound

About device sound

Device sounds refer to audio that is played directly by the device, rather than by the Flash Lite player. Different devices may support different sound formats, including MIDI, MFi, or MP3. To incorporate device sounds in your Flash Lite content you can either include them within your published SWF file, or load external sound files over the network or from the device's local file system. With some exceptions, you can use the ActionScript Sound object to control device sounds, as you would control sounds in the desktop version of Flash Player.

Using bundled device sound

To bundle a device sound in your application, you first import a proxy sound in a format that the Flash authoring tool recognizes, such as an MP3, WAV, or AIFF file. Then you link the proxy sound to a device sound file on your computer that you want to bundle in your application. During the SWF file publishing process, the Flash authoring tool replaces the proxy sound with the linked external sound. During playback, Flash Lite passes the sound data to the device to decode and play.

You can also package multiple device sounds in different formats in a single Flash sound bundle (FLS) file. This is useful if you're creating the same content for several devices that support different device sound formats. For more information, see “[Create a sound bundle](#)” on page 65.

Import and play a device sound

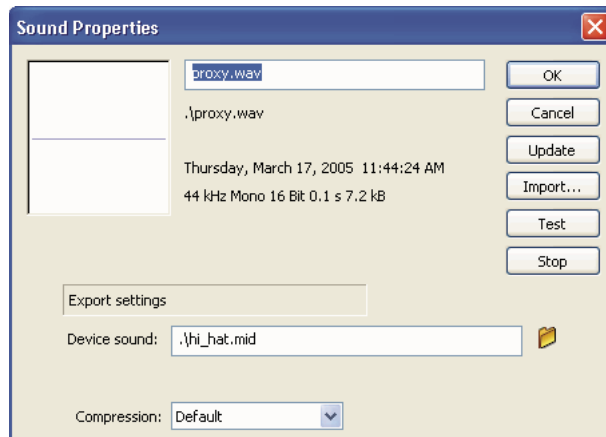
To play a device sound you can either attach it to the Timeline or use the Sound object to play the device sound with ActionScript.

- 1 In Flash, create a new mobile document and save it as **device_sound fla**.
- 2 Select File > Import > Import to Library. On the Samples and Tutorials page at www.adobe.com/go/learn_flt_samples_and_tutorials, locate, download and decompress the ZIP file for your Flash Lite version, and then navigate to the Samples folder.
- 3 Select the proxy.wav file and click OK.

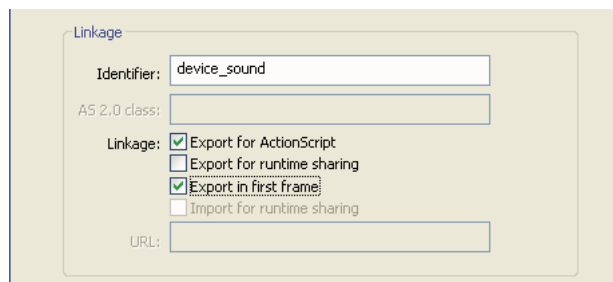
Note: You can use any sound file that's recognized by the Flash authoring tool as the proxy sound. The proxy.wav file is provided for your convenience.

- 1 To link the proxy sound to the device sound file, do the following:
 - a In the Library panel, right-click (Windows) or Control-click (Macintosh) the proxy sound symbol and select Properties from the context menu to open the Sound Properties dialog box.

- b** In the Sound Properties dialog box, click the folder icon to the right of the Device Sound text box to open the Select Device Sound dialog box.
- c** Browse to www.adobe.com/go/learn_flash_samples_and_tutorials. On the Samples and Tutorials page, locate, download and decompress the ZIP file for your Flash Lite version, and then navigate to the Samples folder and select the file named hi_hat.mid.



- d** (Optional) To control the device sound with ActionScript, click Advanced to display the advanced sound properties options, select Export For ActionScript, and type **device_sound** in the Identifier text box.



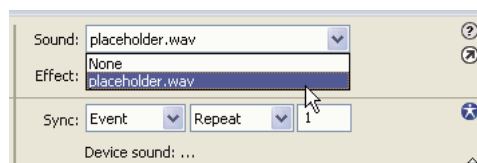
- e** Click OK to close the Sound Properties dialog box.

To play the device sound, you can either attach the proxy sound to the Timeline or use the ActionScript sound object. To use the ActionScript sound object, skip to step 6.

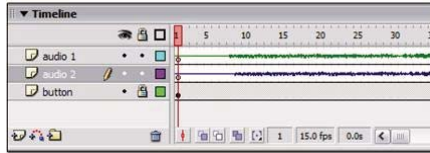
- 2** To attach the device sound to the Timeline, do the following:

- a** Select the keyframe on Frame 1 on the layer named Content.
- b** In the Property inspector, select proxy.wav from the Sound pop-up menu.

This attaches the proxy sound to the keyframe.



The Flash authoring tool displays the proxy sound's waveform in the Timeline. Waveforms for sounds that are linked to external device sounds are colored green; waveforms for sounds that are not linked to external device sounds are colored blue, as the following image shows.



A. Sound linked to external device sound B. Sound not linked to external device sound

- 3 To play the sound with ActionScript, do the following:
- 4 Select the layer named Actions in the Timeline.
- 5 Open the Actions panel (Window > Actions), and type the following code:


```
var deviceSound:Sound = new Sound();
deviceSound.attachSound("device_sound");
deviceSound.start();
```
- 6 Select Control > Test Movie to start the Adobe Device Central emulator and test your SWF file.

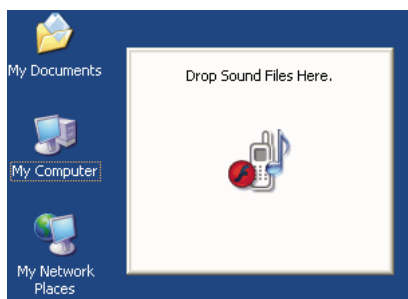
Create a sound bundle

Flash Lite provides the ability to encapsulate several device sounds of different formats into a single *sound bundle*. As an example, a single Flash application can contain the same sound represented in both MIDI and MFi formats. When the application is played on a device that supports MIDI audio, Flash Lite selects the MIDI sound data from the sound bundle and passes it to the device to play. Similarly, if the application is played on a device that supports only MFi, Flash Lite passes the MFi sound data to the device.

You use a utility called the Flash Lite Sound Bundler to create a sound bundle (FLS) file. You then link the FLS file to a proxy sound in your Flash Lite document, just as you would do for a single device sound. For more information about adding device sounds to your Flash Lite applications, see “[Using bundled device sound](#)” on page 63.

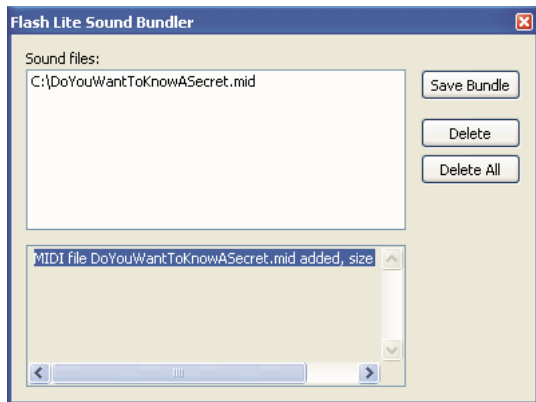
Note: As of this writing, the Sound Bundler utility is supported by Windows systems only.

- 1 Locate and download the Flash Lite Sound Bundler application (FlashLiteBundler.exe) at www.adobe.com/go/developer_flashlite.
- 2 The Sound Bundler appears as a floating window.



- 3 From your desktop, drag the first sound file to be bundled into the Sound Bundler window.

Another window appears that lists the contents of the sound bundle. The lower part of the window contains information about the sounds in the sound bundle, including sound format, size of sound data, and filename.



A. Information about files in sound bundle B. List of files in sound bundle

- 4 Drag the rest of the sound files that you want to bundle into the window.

You can't bundle more than one file in a given audio format. For example, you can't bundle two MIDI files in the same FLS file.

- 5 To delete a file from the sound bundle, select the file in the list and click Delete.

To delete all files in the sound bundle, click Delete All.

- 6 To save the sound bundle, click Save Bundle, and choose a name and location for the FLS file.

- 7 To exit the Sound Bundler, click the close button (X) in the Sound Bundler window.

The next step is to add the sound bundle (FLS) file to your Flash document. The process is the same as adding standard device sounds to Flash documents, except that instead of specifying a single device sound file to replace the proxy sound, you specify the FLS file that you created. For more information, see [“About device sound”](#) on page 63).

Playing external device sounds

In addition to playing device sounds that are bundled in the published SWF file (see [“Using bundled device sound”](#) on page 63), you can also load and play external sound files. To play external device sounds you use the `loadSound()` method of the Sound object. As with bundled device sound, the Flash Lite player passes the externally loaded audio to the device to decode and play.

The following information about playing external device sounds in Flash Lite is important to remember:

- Unlike in the desktop version of Flash Player, externally loaded sounds in Flash Lite are always treated as event sounds. This means that external device sounds do not stream—that is, play as they are being downloaded. The sound data must download fully before you can play the sound. For the same reason, you must call the Sound object's `start()` method to play the sound once it is fully loaded (see the following code example).
- The Flash Lite implementation of the `loadSound()` method does not support that method's second parameter (*isStreaming*). Flash Lite ignores this parameter if it is present.
- Flash Lite does not play externally loaded MP3 files natively. If your application loads an external MP3 file, Flash Lite passes the sound data to the device to decode and play, as it does with any externally loaded sound file.

The following code shows how to load and play an external sound file:

```
// Create the sound object.
var mySound:Sound = new Sound();
// Define onLoad handler for sound,
// which starts the sound once it has fully loaded.
mySound.onLoad = function(success){
    if(success == true) {
        mySound.start();
    }
}
// Load the sound.
mySound.loadSound("http://www.adobe.com/audio.midi");
```

Synchronizing device sounds with animation

Device sounds in Flash Lite are always treated as event sounds. This means that you can't synchronize device sounds with animation in the timeline in the same manner that you can with native Flash sounds. However, you can use device sounds to approximate true synchronized sound by setting the Flash Lite player's `_forceframerate` property to `true`. When this property is set to `true`, Flash Lite drops frames from animation to maintain the SWF file's specified frame rate. As long as the device sound data is authored for the correct duration, and as long as the device plays back sound data at the expected rate, animation and sound will be close to being synchronized.

For example, suppose you have a device sound that's 5 seconds long. During playback, you'd like this sound to play in synch with animation in the timeline. Also assume that your application's frame rate is set to 15 FPS. When you start the sound—either by attaching it to a frame in the timeline or calling `Sound.start()`—you simultaneously set `_forceframerate = "true"`. Subsequently, for each second of device audio playback, Flash Lite ensures that the playback head has advanced 15 frames in the timeline. If, for some reason, the player cannot render each frame in the animation during this time, it drops frames to maintain the specified frame rate.

For more information about the `_forceframerate` property, see `_forceframerateproperty` in the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Determining supported device sound formats

The `System.capabilities.audioMIMETypes` property contains an array of audio MIME types that the device supports. You can use this information to determine what types of device audio your application can play. The array's indexes are the same as the supported MIME types, so you can easily check whether a device supports a specific type of audio.

For example, the following code first checks whether the device supports playback of MIDI audio before loading an external MIDI file:

```
if (System.capabilities.audioMIMETypes["audio/midi"]) {
    my_sound.loadSound("soundtrack.mid");
}
```

Native sound

About native Flash sounds

In addition to supporting device sound, Flash Lite also supports standard, or *native*, Flash sounds. Essentially, a native sound is a sound in any format that the Flash authoring tool recognizes. Native sound in Flash Lite can play either as event sound or synchronized sound (see [“About device sound”](#) on page 63).

The general workflow for using native sounds in your Flash Lite applications is the same as for Flash desktop applications, with the following exceptions:

- Flash Lite does not support playing external MP3 files *natively*. However, if the device can play MP3 audio, you can load external MP3 files in your application. For more information, see “[Playing external device sounds](#)” on page 66.
- Flash Lite does not support the Speech audio compression option.

For more information about working with native sound in Flash, see the “Working with sound” topics in *Using Flash*.

Using the 8 kHz sampling rate feature

By default, the Flash authoring tool exports native audio at sampling rates of 5, 11, 22, or 44 kilohertz (kHz). However, many devices don’t support playback of audio data at these sampling rates. For example, the Nokia Series 60 devices support only 8 and 16 kHz. In previous versions of Flash Lite, the player resampled the audio at runtime to 8 kHz before calling the device’s native sound application programming interfaces (APIs) to play the sound. This resampling required additional processing time and memory consumption.

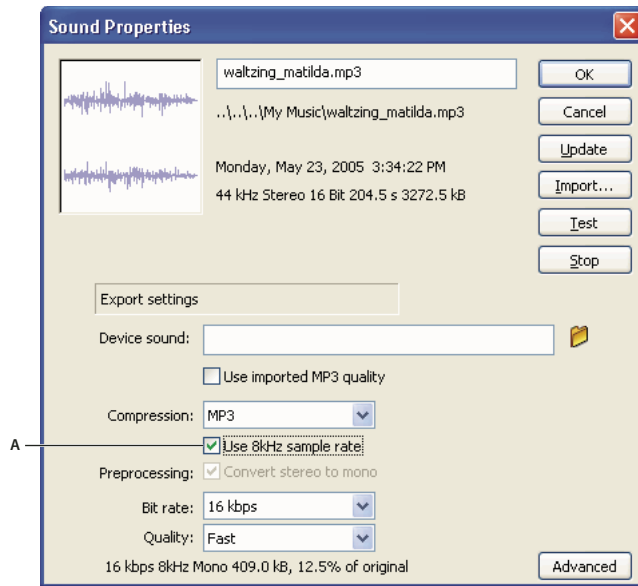
In Adobe Flash CS4 Professional, you can choose to have the authoring tool resample your application’s native audio at 8 kHz during the SWF file publishing process. You apply this setting to all stream and event sounds in your application, or just to specific sounds in your document’s Library panel.

The first procedure that follows explains how to enable the 8 kHz sampling rate option for individual sounds in the Sound Properties dialog box; the second procedure explains how to use the Publish Settings dialog box to set this option globally for all event and stream sounds.

Note: For stream sounds, Flash always applies the global sound compression options that you specify in the Publish Settings dialog box. The per-sound compression options that you specify in the Sound Properties dialog box apply only to event sounds.

Enable the 8 kHz sampling rate feature for an individual sound

- 1 In Flash, right-click (Windows) or Control-click (Macintosh) a sound symbol in the Library panel and choose Properties from the context menu.
- 2 In the Sound Properties dialog box that appears, select MP3 from the Compression pop-up menu.
The 8 kHz sampling feature is available only for MP3-compressed audio.
- 3 Deselect the Use Imported MP3 Quality option, if selected.
- 4 Select the Use 8kHz Sample Rate option.

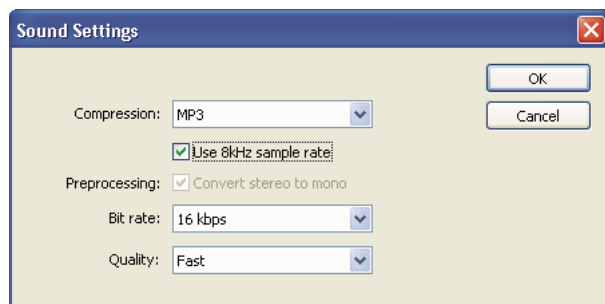


A. Using 8kHz Sample Rate option

- 5 Click OK.

Enable the 8 kHz sampling rate feature globally for all native sounds

- 1 In Flash, select File > Publish Settings.
- 2 In the Publish Settings dialog box, click the Flash tab.
- 3 To enable the 8 kHz sampling rate for all stream sounds in your application, do the following:
 - a Click Set Stream sound options.
 - b In the Sound Properties dialog box that appears, select MP3 from the Compression pop-up menu.
The 8 kHz sampling feature is available only for MP3-compressed audio.
 - c Select the Use 8kHz Sample Rate option.



- d Click OK to close the Publish Settings dialog box.
- 4 To enable the 8 kHz sampling rate for all event sounds, do the following:
 - a Click Set Event Sound Options.
 - b In the Sound Properties dialog box that appears, select MP3 from the Compression pop-up menu.

Note: *The 8 kHz sampling feature is available only for MP3-compressed audio.*

- c** Select the Use 8kHz Sample Rate option.
 - d** Click OK.
- 5** Click OK to close the Publish Settings dialog box.

Chapter 7: Working with video

You can include video in both Flash Lite 2.x and Flash Lite 3.x applications. In Flash Lite 2.x applications, the mobile device decodes and renders the video. This type of video is called *device video*. Flash Lite 3.x applications support two types of video: device video and FLV files (Flash Video). When you play an FLV file in an application, Flash Lite 3.x decodes and renders the video.

Working with FLV files

About FLV file support in Flash Lite 3.0 and later

Flash Lite 3.0 added support for FLV files (Flash Video) using versions of the On2 and Sorenson codecs optimized for mobile devices. Flash Lite 3.x, not the device, renders the FLV files. If a device supports Flash Lite 3.x, you can add FLV files to your application.

Note: *Device manufacturers can choose not to support FLV files if they have already implemented an efficient device video playback mechanism. In this case, use device video in your applications. Check the Device Profiles tab in Adobe Device Central to see whether FLV files (Flash Video) are supported on a particular device.*

You can include FLV files in an application in any of the following ways:

- Embed FLV files in the application.
- Play external FLV files from a local hard drive or web server over HTTP.
- Play external FLV files from Flash Media Server over RTMP.

To convert video (such as QuickTime or Windows Media video files) to FLV files, do any of the following:

- Use an encoder (such as Adobe Flash Video Encoder, or Sorenson™ Squeeze).
- Import the file into the Flash authoring tool and export it as an FLV file.
- Use the FLV Export plug-in to export FLV files from supported video-editing applications.

The following video features are not available in Flash Lite 3.x:

- The Camera class or the recording of video.
- RTMPT and RTMPS connections. These connection attempts default to an RTMP connection.
- Two-way communication between clients. Remote shared objects are not supported. Clients cannot share shared object data.
- Alpha channel video.
- The FLVPlayback component. See [“About components in Flash Lite”](#) on page 1 for information about components.

See also

[“Embed FLV files in an application”](#) on page 72

[“Play external FLV files in an application”](#) on page 72

Embed FLV files in an application

To embed an FLV file, import it into the library of a document. When you publish the application, the authoring tool embeds the FLV file into the published SWF file.

Embedded FLV files are attached to the Timeline. For example, if a Flash document is set to play at 10 frames per second, a 10-second FLV file fills the Timeline from frames 1 to 100. To control playback of an embedded FLV file, use Adobe ActionScript to play, stop, and move the playhead in the Timeline. You can also give the embedded video object an instance name and control it with the properties of the ActionScript Video object.

When you embed FLV files, you can synchronize them with other elements in your document. For example, you could add interactive elements on certain frames that link to other documents. Embedding FLV files is recommended if the files are short and don't have an audio track. Embedding FLV files can add significantly to the size of the published SWF file.

Note: In Adobe Flash Lite 2.0 applications, you can embed device video in an application. This practice is referred to as “bundling video.” See [“Import device video into an application”](#) on page 75.

- 1 In Flash, select File > Import > Import Video.

The Video Import wizard appears.

- 2 Select the option to import a file on your computer and click Browse.

- 3 Browse to the folder that contains the FLV file and select it.

If you don't see the video file listed (or if you can see it but can't select it), select All Files (*.*) from the Files Of Type pop-up menu (Windows), or All Files from the Enable pop-up menu (Macintosh).

- 4 Click Open. In the Import Video wizard, click Next.

- 5 On the Deployment screen, choose Embed Video In SWF And Play In Timeline and click Next.

- 6 On the Embedding screen, select Embedded video as the symbol type.

- 7 Click Next to go to the Finish Video Import screen and click Finish to import the video.

A new video symbol linked to the FLV file appears in the Library panel and on the Stage.

- 8 Select the video object on Stage and enter an instance name for the Video object in the Property inspector.

You can use the instance name to control the Video object with the properties of the ActionScript Video class.

Play external FLV files in an application

An external FLV file is not embedded in a SWF file. An external FLV file resides on the memory card of a device, on a remote web server, or on Adobe Flash Media Server. You can use ActionScript to play and control external FLV files in Flash Lite 3.x at runtime.

Playing external FLV files provides several advantages over embedding video in a Flash document. You can use long video clips without slowing down playback because external FLV files require less memory than embedded video files. External FLV files can have different frame rates than the Flash document in which they play. For example, you can set the Flash document frame rate to 30 frames per second (fps) and the video frame rate to 21 fps. This control ensures smooth video playback. This control also allows you to play FLV files at different frame rates without having to alter existing rich media content.

External FLV files display in Video objects. You can manipulate Video objects much like you can manipulate MovieClip objects. You can rotate and scale the video. You can set the alpha transparency value. You can composite and blend the video with other rich media content. You can also use the Sound class to control the sound of the video.

- 1 In the library in Flash, choose New Video to create a video symbol. Make sure that the video type is “Video (ActionScript-Controlled)” and click OK.
- 2 Drag the video symbol to the Stage and give it an instance name. This example uses the instance name `vo`.
- 3 In the Timeline, select frame one and open the Actions panel.
- 4 Enter the following code :

```
// Create a NetConnection object
var nc:NetConnection = new NetConnection();
// Declare a NetStream object
var ns:NetStream;
/* When a change in connection status occurs,
the onStatus handler is called and passed an
information object with information about the change.
Use the information object to run code when
connection events occur.
*/
nc.onStatus = function(info) {
    // Output all connection status information
    trace(info.code);
    // Run the following code if the connection is successful.
    if(info.code == "NetConnection.Connect.Success"){
        // Create a NetStream object and pass it the NetConnection object
        ns = new NetStream(nc);
        // Attach the NetStream object to the video object on Stage.
        vo.attachVideo(ns);
        // Play the file "sample.flv". Do not include ".flv" when
        // playing the file on Flash Media Server.
        ns.play("sample");
    }
}
/* Call NetConnection.connect() to create a connection.
To connect to a local file or to a file on a web server, pass null.
To connect to Flash Media Server, pass the address to
an application on the server.
*/
nc.connect("rtmp://fmsexamples.adobe.com/vod");
```

Note: To load FLV files from a web server, register the filename extension and MIME type with the web server. The MIME type for FLV files is `video/x-flv`. The MIME type may already be registered with the web server. For more information, check the web server documentation.

Controlling FLV files with ActionScript

Use ActionScript to connect to, display, and control external FLV files. Use the `NetConnection` class to establish a connection. Use the `Video` class to display the video. Use the `NetStream` class to control playback.

To control FLV files, call the methods of the `NetStream` class, for example, the `NetStream.pause()` method. To control device video, call the methods of the `Video` class. For example, to pause device video, call the `Video.pause()` method.

See the *Flash Lite 2.x and 3.x ActionScript Language Reference* for detailed descriptions of these classes, along with code examples that illustrate their use. In addition, Flash Lite 3.x supports all the methods, properties, and events documented in the *ActionScript 2.0 Language Reference*, except for the `MovieClipLoader.checkPolicyFile` property.

Working with device video

About device video

Beginning with version 2.0, Flash Lite can play *device video*, which refers to any video format or encoding supported by the target device. Devices support different video codecs and formats. Some common device video formats are 3GP, 3G2 (or 3GPP2), and MPEG-4. Flash Lite can play any video format that the target device supports.

Flash Lite does not decode or render device video. Instead, Flash Lite relies on the device to decode and render device video. For this reason, device video has the following limitations:

- You cannot rotate or skew device video. Some devices support scaling.
- You cannot synchronize device video with the Timeline.
- You cannot composite or blend device video with other media. The device renders video on top of any other rich media content.
- You cannot control the sound volume of a video clip.

To deploy device video, bundle it in an application or load it from an external file on the device or from a network location.

See also

[“Determining supported device video formats”](#) on page 74

[“Import device video into an application”](#) on page 75

[“Play external device video in an application”](#) on page 79

Determining supported device video formats

Different devices support different video formats and encodings. To determine which formats a device supports, do one of the following:

- Use the Adobe Device Central Device Profiles tab. For more information, see Device Central Help.
- See the specifications of the device manufacturer.
- Use the ActionScript `System.capabilities.videoMIMETypes` property.

The `System.capabilities.videoMIMETypes` property contains an array of video MIME types that the device supports. Each item in the array has the format `video/video-type`. For example, the following code displays all of the video MIME types that the device supports in the Output window:

```
var mimeTypes = System.capabilities.videoMIMETypes;  
trace(mimeTypes.toString());
```

Note: To view the Output window in Adobe Device Central, choose *View > Flash Output > Show*.

The following code checks whether a device supports 3GPP video before playing a video file of that type:

```
if (System.capabilities.videoMIMEtypes["video/3gpp"]) {
    my_video.play("movie.3gp");
}
```

Import device video into an application

To bundle device video, import it into the library of a Flash document. When you publish the application, the Flash authoring tool embeds the FLV file into the published SWF file.

To import a video file into a Flash document, use the Import Video wizard or the Library panel. Both techniques add a video symbol to the document library.

Use the Import Video wizard

- 1 In Flash, select File > Import > Import Video.
- 2 Select the option to import a file on your computer and click Browse.
- 3 Browse to the folder that contains the device video file and select it.
If you don't see the desired video file (or if you can see it but can't select it), select All Files (*.*) from the Files Of Type pop-up menu (Windows), or All Files from the Enable pop-up menu (Macintosh).
- 4 Click Open.
- 5 In the Import Video wizard, click Next.
- 6 For Flash Lite 2.0 applications, the only deployment option is As mobile device video bundled in SWF. Click Finish.
A video symbol appears in the Library panel. This symbol is associated with the device video file.
- 7 Drag the video object from the Library to the Stage and give it an instance name in the Property inspector.

Use the Library panel to import device video

- 1 In Flash, open the Library panel (Window > Library).
- 2 Open the Library options menu and choose New Video.
The Video Properties dialog box appears.
- 3 In the Video Properties dialog box, select the option to bundle the source video in the SWF file, then click Import.
- 4 Browse to the folder that contains the device video file and select it.
If you don't see the desired video file (or if you can see it but can't select it), select All Files (*.*) from the Files Of Type pop-up menu (Windows), or All Files from the Enable pop-up menu (Macintosh).
- 5 Click Open.
- 6 In the Video Properties dialog box, select the Export For ActionScript option and enter a string in the Identifier box.
- 7 Click OK to close the Video Properties dialog box.
A video symbol appears in the Library panel. This symbol is associated with the device video file.

Play bundled device video in an application

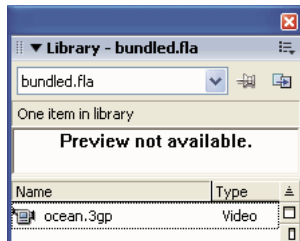
To create this application, drag a video to the Stage and add ActionScript to buttons to play and stop the video.

- 1 In Flash, choose File > New > Flash File (Mobile) and click OK.
- 2 In Adobe Device Central, click Create.
- 3 In Flash choose File > Save and save the file as bundled_video fla.

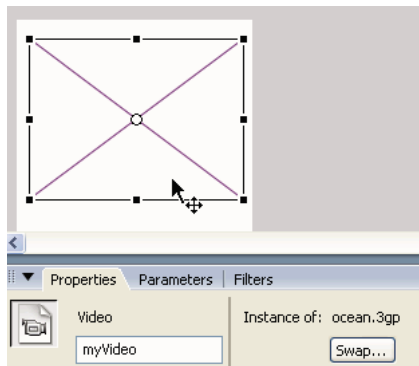
- 4 Import the device video file ocean.3gp from the Flash_Lite_2x\Samples\Video\ folder. For more information about importing a device video, see “Import device video into an application” on page 75.

Note: On the Samples and Tutorials page (www.adobe.com/go/learn_flt_samples_and_tutorials), download and decompress the ZIP file for your Flash Lite version.

A new video symbol appears in the Library panel.



- 5 Drag an instance of the video symbol to the Stage.
- 6 Select the video object on the Stage and, in the Property inspector, type **myVideo** in the Instance Name box. Use this name to refer to the video object in ActionScript code.



- 7 In the Property inspector, set the width to 176 and the height to 144. These dimensions match the source video. Depending on the device, a device video in Flash Lite does not always scale to fit the size of the bounding box.
- 8 To add buttons to control the video, open the library of prebuilt buttons (Window > Common Libraries > Buttons).
- 9 In the Buttons library, double-click the Circle Buttons folder to open it.
- 10 Drag an instance of the Play button symbol to the Stage.
- 11 Drag an instance of the Stop button symbol to the Stage.
- 12 Select the Play button on the Stage and enter (or copy and paste) the following code into the Actions panel:

```
on(press) {  
    myVideo.play();  
}
```

- 13 Select the Stop button on the Stage and enter the following code in the Actions panel:

```
on(press) {  
    myVideo.stop();  
}
```


14 Choose Control > Test Movie to test the movie in Device Central.

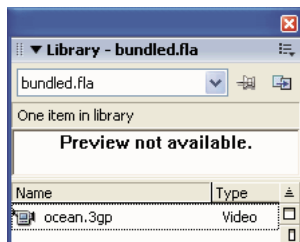
Play bundled device video in an application at runtime

To create this application, drag a video placeholder to the Stage. Add ActionScript to buttons to play and stop the video. The Play button pulls the video dynamically from the Library panel at runtime. This application uses only one video. An application can have any number of video files in the Library panel and pull them into the video placeholder at runtime.

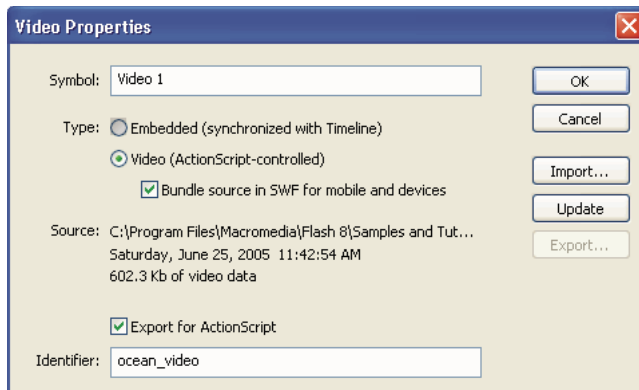
- 1 In Flash, choose File > New > Flash File (Mobile) and click OK.
- 2 In Adobe Device Central, click Create.
- 3 In Flash choose File > Save and save the file as `bundled_video2.fla`.
- 4 Import the device video file `ocean.3gp` from the `Flash_Lite_2x\Samples\Video\` folder. For more information about importing a device video, see “[Import device video into an application](#)” on page 75.

Note: On the Samples and Tutorials page (www.adobe.com/go/learn_flt_samples_and_tutorials), download and decompress the ZIP file for your Flash Lite version.

A new video symbol appears in the Library panel.

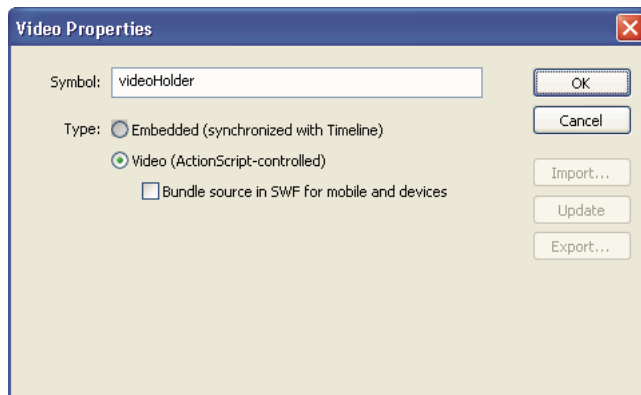


- 5 Right-click (Windows) or Control-click (Macintosh) the `ocean.3gp` video symbol in the library and choose Properties from the context menu. The Video Properties dialog box appears.
- 6 In the Video Properties dialog box, select Export For ActionScript and enter **ocean_video** in the Identifier box, as the following image shows:



- 7 Click OK to close the Video Properties dialog box.
- 8 To create the placeholder video clip, do the following:
 - a In the Library panel, click the options menu button in the title bar and select New Video. The Video Properties dialog box appears.

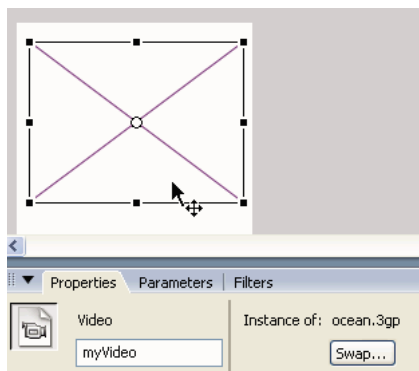
- b** In the Symbol box, type **videoHolder**.



- c** Click OK to close the Video Properties dialog box.

- 9** Drag an instance of the video symbol to the Stage.

- 10** Select the video object on the Stage and, in the Property inspector, type **myVideo** in the Instance Name box.
Use this name you to refer to the video object in ActionScript code.



- 11** In the Property inspector, set the width to 176 and the height to 144.

These dimensions match the source video. Depending on the device, a device video in Flash Lite does not always scale to fit the size of the bounding box.

- 12** To add buttons to control the video, open the library of prebuilt buttons (Window > Common Libraries > Buttons).

- 13** In the Buttons library, double-click the Circle Buttons folder to open it.

- 14** Drag an instance of the Play button symbol to the Stage.

- 15** Drag an instance of the Stop button symbol to the Stage.

- 16** Select the Play button on the Stage and enter (or copy and paste) the following code into the Actions panel:

```
on (press) {
    myVideo.play("symbol://ocean_video");
}
```

- 17** Select the Stop button on the Stage and enter the following code in the Actions panel:

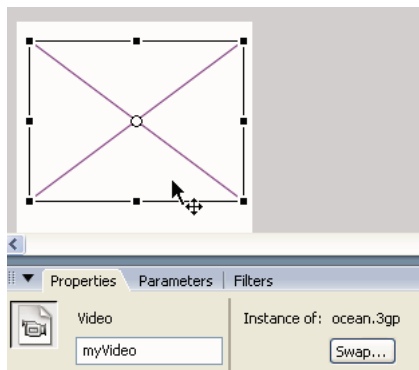
```
on (press) {
    myVideo.stop();
}
```

18 Choose Control > Test Movie to test the movie in Device Central.

Play external device video in an application

External device video files reside on the memory card of a device or on a web server. As with all device video, the device is responsible for decoding and displaying the files.

- 1 In Flash, choose File > New > Flash File (Mobile) and click OK.
- 2 In Adobe Device Central, click Create.
- 3 In Flash choose File > Save and save the file as external_device_video.fla.
- 4 Save the file ocean.3gp to the same folder as the external_device_video.fla file.
The file is located in the Flash_Lite_2x\Samples\Video\ folder in the Samples and Tutorials package you can download from www.adobe.com/go/learn_flt_samples_and_tutorials.
- 5 To create the video symbol to display the external file, do the following:
 - a In the Library panel, click the options menu button in the title bar and select New Video. The Video Properties dialog box appears.
 - b In the Symbol box, enter **Video Display**.
This name is used in the Library panel.
 - c Select Video (ActionScript-controlled) and click OK.
Make sure Bundle source in SWF for mobile and devices is unchecked.
- 6 Drag an instance of the video symbol to the Stage.
- 7 Select the video object on the Stage and, in the Property inspector, type **myVideo** in the Instance Name box.
Use this name to refer to the video object in ActionScript code.



- 8 In the Property inspector, set the width of the video object to 176 and the height to 144.
These dimensions match the source video. Depending on the device, a device video in Flash Lite does not always scale to fit the size of the bounding box.
- 9 To add buttons to control the video, open the library of prebuilt buttons (Window > Common Libraries > Buttons).
- 10 In the Buttons library, double-click the Circle Buttons folder to open it.

11 Drag an instance of the Play button symbol to the Stage.

12 Drag an instance of the Stop button symbol to the Stage.

13 Select the Play button on the Stage and enter (or copy and paste) the following code into the Actions panel:

```
on (press) {
    myVideo.play("ocean.3gp");
}
```

To play an external video file, pass the absolute or relative file location or URL of the video file to the `video.play()` method. In the following example, the SWF file and the 3GP file are located in the same folder on the device. You can also specify a folder path relative to the SWF file, as follows:

```
myVideo.play("folder1/folder2/ocean.3gp");
```

Depending on the device, you can also use the `file://` protocol to play a video file at a specific location, as follows:

```
myVideo.play("file:///c:/folder1/folder2/ocean.3gp");
```

Note: *Not all devices support the `file://` protocol. Be sure to test your application on all target devices if you use this protocol.*

Some devices support loading video files from network addresses over the RTSP (Real Time Streaming Protocol). HTTP-based streaming is not supported. The following example loads and plays a 3GPP file:

```
myVideo.play("rtsp://www.example.com/video/ocean.3gp");
```

14 Select the Stop button on the Stage and enter the following code in the Actions panel:

```
on (press) {
    myVideo.stop();
}
```

15 Choose Control > Test Movie to test the movie in Device Central.

View and edit device video symbol properties

You use the Video Properties dialog box to view and edit information for video symbols in the Library panel.

Open the Video Properties dialog box

Do one of the following:

- Right-click (Windows) or Control-click (Macintosh) a video symbol in the Library panel and choose Properties from the context menu.
- Select a video symbol in the Library panel and choose Properties from the options menu in the title bar.

Import a device video into a video symbol

- 1 Select a video symbol in the Library panel and open the Video Properties dialog box.
- 2 In the Video Properties dialog box, select the option to bundle the video source in the SWF file, if it's not already selected.
- 3 Click Import and, in the file browser, locate the device video file to import and select it.
If you don't see the desired video file (or if you can see it but can't select it), select All Files (*.*) from the Files Of Type pop-up menu (Windows), or All Files from the Enable pop-up menu (Macintosh).
- 4 Click Open to close the file browser.
- 5 Click OK to close the Video Properties dialog box.

Note: The OK button is dimmed until you either import a device video or deselect the option to bundle the video source in the SWF file.

Assign an identifier to a video symbol

- 1 Select the video symbol and open the Video Properties dialog box.
- 2 Import a device video into the symbol, if you haven't already.
- 3 Select Export For ActionScript.
- 4 In the Identifier box, enter an identifier for the video symbol.

Like all identifiers in ActionScript, the first character must be a letter, underscore (_), or dollar sign (\$). Each subsequent character can be a number, letter, underscore, or dollar sign.

- 5 Click OK to close the Video Properties dialog box.

Controlling device video with ActionScript

Use the ActionScript Video object to control device video playback. The following methods of the Video object are available in Flash Lite 2.0 and later:

- `Video.play()`
- `Video.stop()`
- `Video.pause()`
- `Video.resume()`
- `Video.close()`
- `Video.attachVideo()` (Flash Lite 3.0 and later)

Note: These methods of the Video object correspond roughly to the same methods available in the NetStream object in the desktop version of Flash Player. These methods of the Video object are not available in the desktop version of Flash Player.

Flash Lite 2.0 and later do not support the following methods and properties of the Video object:

- `Video.clear()`
- `Video.deblocking`
- `Video.height`
- `Video.smoothing`
- `Video.width`
- `Video._visible`

For more information about using the Video object in Flash Lite, see the Video object entry in *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Chapter 8: Developing Flash Lite applications for BREW

You can develop Adobe Flash Lite 2.x applications that can run on devices that use the Binary Runtime Environment for Wireless (BREW) platform made by QUALCOMM Incorporated. If you are a developer who is comfortable working with Flash Lite but are not familiar enough with the BREW platform and its requirements to readily produce applications for BREW devices, you should read this chapter. You will also find this chapter useful if you have expertise with other application development technologies and want to explore using Flash Lite to develop BREW applications.

BREW basics

About BREW

The QUALCOMM BREW mobile device platform is installed and supported on a variety of devices made by different manufacturers worldwide. Adobe Flash CS4 Professional includes tools and resources that developers use to create Flash Lite-based applications and content that users can download and use on BREW devices.

Flash developers use the Flash authoring tool to create Flash Lite content and applications for BREW-enabled devices. Developers then submit their applications to National Software Testing Laboratories (NSTL) for True BREW Testing (TBT). Once applications pass TBT, they are uploaded to the BREW Delivery System (BDS), and if chosen for distribution by a carrier, they are made available on the carrier's application download server (ADS). Device users can then download and purchase these applications, taking advantage of key features in the BREW platform for over-the-air distribution and billing through the BDS.

Developers who want to experiment with BREW without incurring the costs associated with full-scale BREW development can elect not to register as authenticated BREW developers and download only the free SDK (Software Development Kit) components, which are sufficient to produce and test an application using the BREW Simulator, but not to upload and test it on a device, or to simulate specific device characteristics with customized device packs.

BREW is intended to operate as an “ecosystem” to help support the development and delivery of wireless device content. Its aim is to make it easier for developers to create, distribute, and derive profit from wireless applications. This ecosystem consists of the following main groups of participants:

Developers use tools available in the BREW SDK to develop content with C++ combined with APIs, or with a BREW-compatible extension. The BREW SDK is a free download, but you must register with QUALCOMM to get access to the Tools Suite and SDK Tools.

National Software Testing Laboratories (NSTL) tests applications that developers submit, and if they pass, the applications are eligible for inclusion in the BDS. If they do not pass, developers can correct and resubmit their applications.

QUALCOMM maintains the BREW Delivery System (BDS), which is a web-based system that delivers content listed in QUALCOMM's catalog, along with pricing and billing information. Pricing can involve several models, including operator auction.

Operators (carriers) use the application download server (ADS) to offer content they select at retail prices for purchase by customers through an over-the-air (OTA) delivery mechanism.

Device manufacturers use the BREW client, which exposes a common set of application programming interfaces (APIs) for standardized development of wireless applications. The client also includes an application manager that users can purchase and use to manage BREW applications.

Flash Lite and BREW development tools

To develop and test Flash Lite applications for the BREW environment, you need to acquire and install specific hardware and software tools.

Hardware

Windows computer For information about system requirements for the BREW SDK and the BREW tools suite, see the “System Requirements” section in *Starting with BREW* (available from the BREW website). For information about system requirements for Adobe software, see www.adobe.com/go/sysreqs.

BREW-enabled devices with test bit set For information about how to acquire, activate, and test-enable BREW devices, see the “Handset Acquisition and Readiness” section of *Getting Started with BREW* (available from the BREW website). Note that devices that support BREW 2.x must be sent to QUALCOMM to be test-enabled; BREW 3.x devices are pre-enabled.

Data cables Data cables are required to upload Flash applications to the BREW devices you plan to support. Most devices come bundled with data cables, but if you do not have a cable, see the “Acquire BREW Handsets and Data Cables” section of *Getting Started with BREW* (available from the BREW website).

Software

BREW SDK and BREW Tools Suite For information about installing the BREW SDK and tools, see Chapter 2, “Installing the BREW SDK,” in *Getting Started with BREW* (available from the BREW website). You must be a registered BREW developer to download and install the BREW Tools Suite, but not the SDK itself. For instructions on how to register, see the BREW website.

USB drivers for targeted devices Obtain and install the USB drivers for the devices you intend to target. For information about how to acquire the drivers, contact the device manufacturer.

Flash Lite extension for BREW-enabled devices These devices are available from the BREW website for testing purposes only. The extension is downloaded automatically to users’ devices when they select and download a Flash Lite for BREW application.

These components are described in detail in “[Setting up your system for BREW](#)” on page 85.

Flash Lite features supported in BREW

Flash Lite for BREW supports a subset of the features available in both Flash Lite 2.0 and Flash Lite 2.1. The features that Flash Lite for BREW supports are as follows. For further information about these features, see “[Flash Lite 3.0 features](#)” on page 7.

- Built on Macromedia Flash Player 7 from Adobe, supports Adobe ActionScript 2.0
- XML data handling
- Persistent data management
- Device, vector fonts
- Device sound for events
- Inline video (restricted to the video formats that are supported on the device)
- Streaming video

- XML sockets
- Network access and HTTP streaming
- Inline text support (limited to 64,000 characters, with further limitations imposed by individual device implementations. Some devices support as few as 1000 characters.)
- Dynamic multimedia
- Keypad and keyboard support

Flash Lite features not supported in BREW

Manufacturers of BREW devices can limit the use of some Flash Lite features, which means that these features cannot be used on the device. Information about specific devices is available in the BREW website's Developer Resources area.

In addition to features that are not supported on a given device, the following list describes the functionality that is not supported in the current Flash Lite BREW implementation (and is thus not available on any device):

Flash Lite sound decoding All sound formats are passed to the device for playback without modification. For example, if the sound is in mp3 format and the device supports mp3, it plays. Flash Lite does not decode any sounds internally in BREW, as it does on other platforms.

Streaming sound All sound must be fully loaded before it can play. The current BREW implementation does not support playing sounds progressively as they are loading over a network.

ActionScript commands The following ActionScript `fscommand2()` functions are unavailable on any BREW device:

<code>ExtendBacklightDuration()</code>	<code>GetNetworkConnectStatus()</code>
<code>GetBatteryLevel()</code>	<code>GetNetworkName()</code>
<code>GetMaxBatteryLevel()</code>	<code>GetNetworkRequestStatus</code>
<code>GetMaxSignalLevel()</code>	<code>GetPowerSource()</code>
<code>GetNetworkConnectionName()</code>	<code>GetSignalLevel()</code>

The following ActionScript `fscommand()` function is unavailable on any BREW device:

<code>fscommand(Launch)</code>	
--------------------------------	--

getURL() support Some BREW handsets support some `getURL()` functions, depending on the implementation. As a general rule, neither MMS (Multimedia Messaging Service) nor HTTP is supported (no API exists to enable Flash Lite to launch a browser).

Note: *mailto requests are not directly supported. Instead, you can use the Short Message Service (SMS) protocol, which limits the maximum message size to 160 characters.*

Wallpaper content Flash Lite wallpaper content is not supported in the Flash Lite for BREW implementation.

Animated ring tones Animated ring tones are not supported in the Flash Lite for BREW implementation.

BREW SDK versions supported

Flash Lite 2.1 for BREW Devices currently supports BREW versions 2.1.3 and later. You can find the version of BREW that is running on a particular device in the Device Specifications list on the BREW website. The two devices that currently support Flash Lite for BREW (the Samsung SCH-A950 and the LG VX9800) support different versions of BREW (2.x and 3.x, respectively). Because the BREW SDKs are backward-compatible, you can download the SDK and tools for the most recent version (3.x at this writing) rather than the version supported on the device you are targeting for development. For example, if you are targeting the Samsung SCH-A950, which is a 2.x device, you can still download and use the 3.x version of the BREW SDK and Tools Suite.

The differences between BREW 2.x and 3.x are minimal in terms of the user interface, and Flash Lite behaves the same on both platforms. The significant differences are in the way files are stored on the device and in how applications in the BREW Tools Suite are loaded. For more information about file system structure, see “[Device file structures for different BREW versions](#)” on page 92.

Devices supported

The devices that currently support the BREW platform are:

Device	Version supported
Samsung SCH-A950	BREW version 2.1.3
LG VX9800	BREW version 3.1.2

The Adobe customer support team will certify additional devices as they become available. For a current list of supported devices, see www.adobe.com/go/mobile_supported_devices/.

Additional resources

For information about BREW, see the BREW website. The Developer Home area of the site contains information about the BREW SDK, BREW tools and utilities, and the characteristics of specific devices that support BREW.

Note: You must be a registered BREW developer to download and use the BREW Tools Suite, as well as to enable your devices for testing BREW. Details about how to register are available on the site.

Setting up your system for BREW

Workflow for setting up your system for BREW

To set up your system to produce Flash Lite files for BREW, you need to locate and install the appropriate software from several sources. You also need to acquire the appropriate hardware (BREW-enabled devices, data cables, and so on). Much of the specific information required to install and configure these components is located in other Flash Lite or BREW documents.

To set up your system to support Flash Lite authoring for BREW, you complete the following tasks:

- 1 Register as a BREW developer.
- 2 Install the BREW SDK and Tools Suite.
- 3 Install Flash Lite 2.1 for BREW.
- 4 Install USB drivers for target devices.

Register as a BREW developer

To register as a BREW developer, go to the QUALCOMM's main BREW site and follow the links on the Developer home page to become an authenticated BREW developer.

Install the BREW SDK and tools

The instructions that follow describe how to install the BREW SDK on your computer. They assume that you have already registered as a BREW developer. For more detailed information about installing the BREW SDK or Tools Suite, see "Installing the BREW SDK" in *Starting with BREW*. Go to the BREW website and then choose Developer Home > BREW Documentation > Application Development Documentation.

- 1 From BREW Developer Home on the BREW website, click Download BREW Tools to display the BREW Development and Commercialization Tools page.
- 2 Navigate to the BREW Development area and click the appropriate link to download the most recent BREW SDK version (at this writing, BREW SDK 3.1).

Follow the onscreen instructions to download and install the SDK.

- 3 After you install the SDK, install the tools.

Navigate to the BREW Commercialization area of the BREW Development and Commercialization Tools page and click the link for the BREW Testing and Commercialization Utilities. Follow the onscreen instructions to install the Tools Suite on your computer.

Your Start menu should now contain entries for both the BREW SDK and the BREW Tools Suite.

Install Adobe Flash Lite 2.1 for BREW Devices

Flash Lite 2.1 for BREW Devices includes three components:

- Flash Lite Publisher for BREW (available from www.adobe.com/go/support_flashlite)
- Flash Lite 2.1 for BREW Simulator (available from www.adobe.com/go/support_flashlite)
- Flash Lite extension for BREW-enabled devices (available from the BREW website for testing purposes only. The extension is downloaded automatically to users' devices when they select and download a Flash Lite for BREW application.)

Installing USB drivers for devices

Each device you intend to target for BREW development includes its own set of USB drivers, which must be installed on your computer. New devices should include a CD-ROM that contains these drivers; use these versions whenever possible. To install USB drivers for a device, follow the standard procedure for installing device drivers in Windows.

Additional resources

For more information about Flash Lite application development and testing, see the other sections in this document, which provide comprehensive information about how to develop and test Flash Lite applications. The online Help for Flash CS4 Professional also contains extensive information on Flash Lite development.

For more information about BREW, the best source of information is the QUALCOMM BREW website.

You can also consult the Help files included in the SDK for detailed information about BREW development processes and tools.

At this writing, the BREW documentation set includes two similarly named documents, both of which are useful for learning BREW basics. These documents are:

- *Getting Started with BREW* (a two-page summary of the hardware and software you need to develop BREW applications).
- *Starting with BREW* (a more comprehensive document that contains all of the information you need to start developing BREW applications).

Authoring Flash Lite files for BREW

Workflow for authoring Flash Lite files for BREW

Authoring for BREW is similar to authoring for Flash Lite in general, except that you tailor your applications to the characteristics of the specific BREW-enabled handset for which you are designing. This section describes how to use Flash to author Flash Lite content that can later be published for BREW by using the Flash Lite Publisher for BREW (described in “[Publishing Flash Lite files for BREW](#)” on page 89).

Creating Flash Lite content for the BREW platform is an iterative process that involves the following tasks:

Identify your target devices and Flash Lite content type

At this writing the following two devices support BREW: the Samsung SCH-A950, which includes the BREW version 2.1.3 platform; and the LG VX9800, which includes the BREW version 3.1.2 platform. These devices have substantially different characteristics (for example, the LG has a QWERTY keyboard and the Samsung does not), so it is important to tailor your application development plans to the device’s unique capabilities.

Also, you can create two different content types—applications and screen savers; each content type requires different application design concerns. For more information about how screen savers differ from applications, go to the BREW website and navigate to Developer FAQs > BREW Tools > MIF Settings > Screensaver.

Note: *Currently only the Samsung SCH-A950 supports screen saver content types.*

Create and test your application in Flash

Adobe Flash CS4 Professional includes an emulator on Adobe Device Central CS4 that lets you test your application without having to transfer it to a device. You use the Device Central emulator to refine your application’s overall design and fix any problems before you test it on a mobile device.

Note: *The testing you do in Flash using the emulator is different from the testing you do using the Simulator that is included with the BREW SDK. Flash emulator testing is described in this section, and BREW Simulator testing is described in “[Publishing Flash Lite files for BREW](#)” on page 89. Because the Device Central emulator does not currently provide device packs for BREW devices, the Device Central emulator can be used only to test basic functionality. Use the BREW Simulator to test your application’s functionality in the BREW environment.*

Identifying your target device and content type

The QUALCOMM Developer Home contains information about the specific characteristics of all of the devices that currently support a version of BREW. Some information is also available about precommercial devices that are not yet available on the market. You can use this information to determine the devices that you plan to target with your BREW application. Several sample applications with different device packs are also included with the BREW SDK. You can use these sample applications to learn more about how specific devices work.

Creating your application in Flash

The process for creating Flash Lite applications for BREW is similar to creating a generic Flash Lite application. With the exception of features that are not supported on BREW devices (listed in “[Flash Lite features not supported in BREW](#)” on page 84) and BREW-specific authoring notes (below), developing applications for BREW devices can be done by following the development steps detailed in the rest of the Flash documentation.

The following topics provide information that applies only to Flash Lite authoring for BREW.

Launching a browser

The BREW target handsets use the OpenWave browser, which doesn’t provide the appropriate hook for the BREW application to invoke the browser. Therefore, if you specify an HTTP URL in the `getURL()`, `loadVars.send()`, or `XML.send()` functions, no browser is launched. To send information to a server and continue playing your SWF file without opening a new window or replacing content in a window or frame, use the `loadVars.sendAndLoad()` function.

Default soft key behavior

For information and examples describing how to program soft key behavior in Flash Lite, see “[Using the soft keys](#)” on page 47. The following table describes the default soft key behavior in Flash Lite 2.1 for BREW:

Content action	Screen mode	Left key action	Right key action
Default (content does not call the <code>SetSoftKeys</code> command)	Full-screen mode	Displays options menu	Exits the player
Non-full-screen mode (content does not call <code>SetSoftKeys</code>)	Non-full-screen mode	Displays options menu	Exits the player
Content disables soft keys	n/a	None	None
Content overrides one or both soft keys (calls <code>SetSoftKeys</code>)	n/a	Displays options menu if no override. If override, behavior specified by content.	Exits the player if no override. If override, behavior specified by content.

Test your application in Flash

As with creating your application, using the Adobe Device Central emulator to test your BREW application is similar to testing a generic Flash Lite application. Remember that the testing you do in Flash (using the Device Central emulator) is different from the testing you do using the Simulator that is included with the BREW SDK. Device Central emulator testing is described in this section, and BREW Simulator testing is described in “[Publishing Flash Lite files for BREW](#)” on page 89.

Because the Device Central emulator does not currently provide device packs for BREW devices, the Device Central emulator can be used only to test basic functionality. Use the BREW Simulator to test your application’s functionality in the BREW environment.

- 1 Open your application in Flash.
- 2 Select File > Publish Settings. On the Flash tab, select Flash Lite 2.1 from the Version list, and then select ActionScript 2.0 from the ActionScript version list. Click OK to save your settings.
- 3 Launch the Adobe Device Central emulator (Control > Test Movie) and click the Device Profiles tab.

- 4 In the library pane, do one of the following:
 - a If you have not downloaded a device pack for your target device, select Generic > Flash Lite 2.0 > Generic Phone.
 - b If you downloaded a device pack for your targeted device (available from www.adobe.com/products/flash/download/device_profiles), click the device name to expand it, select the appropriate device size, and then drag the name of your targeted device from the Available Devices pane to the Device Sets pane.
- 5 Select File > Return to Flash, and then select Control > Test Movie on the Flash menu. If the BREW Publisher wizard appears, click Cancel to close the dialog box. Flash exports the application to Device Central and opens it in the emulator.
- 6 Use the emulator to test your application.

For information about how to use the emulator, see “Using the emulator” on page 103.

When you have finished testing your application in the emulator, save your files and proceed to “Publishing Flash Lite files for BREW” on page 89 for instructions on how to publish your application for the BREW platform.

Additional resources

For further information about developing and testing Flash Lite applications, consult the following sources, which you can find in Flash Help:

- *Getting Started with Flash Lite 2.x and 3.0* (PDF)
- Testing Flash Lite Content (Help topics)

Publishing Flash Lite files for BREW

About publishing Flash Lite files for BREW

To run on BREW devices, Flash Lite applications must conform to the BREW standard. The Flash Lite for BREW Publisher wizard makes it easier for developers to create applications that conform to this standard by automating some of the steps required. In addition to generating the files for the application itself, the wizard also uses the Simulator tool in the BREW SDK to generate files that you can use to test your BREW application.

Before you start using the BREW Publisher wizard, it is useful to understand some basic information about BREW application file types, file structure, and requirements.

File types

The primary purpose of the Flash Lite Publisher for BREW is to publish a standard SWF file, and then use a post-processor to package the SWF file so that it can run on a BREW device. The following table lists the files needed to use Flash Lite content on a BREW device:

File	File content
MIF file	Configuration information
MOD file	Application file for BREW device
SIG file	Device-specific file from QUALCOMM

The files needed to use Flash Lite content with the BREW Simulator are as follows:

File	File content
MIF file	Configuration information
DLL file	Application file for BREW Simulator

All of these files must have the same filename (plus the appropriate file extension: .mif, .mod, .sig, or .dll), and the filename must also be the same as the application folder name. These files must also be located in the correct folders on the device in order to work. The folder structure varies according to the version of BREW your target device uses. For more information, see “[Device file structures for different BREW versions](#)” on page 92.

MIF files

MIF files are configuration files that provide information required by BREW applets. The BREW Publisher wizard generates MIF files for you as part of the publishing process. However, it is the responsibility of the individual content developer to understand the MIF format and requirements, and to verify that the MIF file settings are correct before submitting applications for True BREW Testing.

The following code shows the default MIF values that are output by the BREW Publisher wizard, for applications and screen savers, respectively. If any of these values are incorrect for your application, download the MIF Editor tool from the BREW website and edit the files before submitting your applications for testing.

```
<Applet>
- List of Applets defined in this Module: [Class ID in post processor]
- Applet Information:
* Name: [Applet Name in post processor]
* Class ID: [Class ID in post processor]
* Applet Type: [blank]
- Graphics
* Large: [Large Icon in post processor]
* Medium: [Medium Icon in post processor]
* Small: [Small Icon in post processor]
<Extension>
- Exported Classes: [blank]
- Exported MIME Types: [blank]
<Dependencies>
- External Classes Used by this Module: [blank]
- External Classes Directory (Checkbox): [checked]
- Used: [Class IDs generated by post processor]
<Privileges>
File [checked]
Network [checked]
Web Access [checked]
TAPI [checked]
<Access Control>
- ACL type [blank]
- Rights [blank]
- Groups [blank]
<License>
No License (Checkbox): [checked]
<General>
- Author Name: [Author in post processor]
- Copyright String: [Copyright in post processor]
- Module Version: [Version in post processor]
- EFS Restrictions
* Max Files: [blank]
```

```
* Max Space: (bytes) [blank]
- String Encoding Format: [ISOLATIN1]
Default MIF output for Screensavers
<Applet>
- List of Applets defined in this Module: [Class ID in post processor]
- Applet Information:
* Name: [Applet Name in post processor]
* Class ID: [Class ID in post processor]
* Applet Type: [blank]
- Graphics
* Large: [Large Icon in post processor]
* Medium: [Medium Icon in post processor]
* Small: [Small Icon in post processor]
- Notifications, Flags, Settings...
* Flags: Screensaver [checked]
<Extension>
- Exported Classes: [blank]
- Exported MIME Types: [brew/ssaver Class ID]
<Dependencies>
- External Classes Used by this Module: [blank]
- External Classes Directory (Checkbox): [checked]
- Used: [Class IDs generated by post processor]
<Privileges>
All unchecked
<Access Control>
- ACL type [blank]
- Rights [blank]
- Groups [blank]
<License>
No License (Checkbox): [checked]
<General>
- Author Name: [Author in post processor]
- Copyright String: [Copyright in post processor]
- Module Version: [Version in post processor]
- EFS Restrictions
* Max Files: [blank]
* Max Space: (bytes) [blank]
- String Encoding Format: [ISOLATIN1]
```

SIG files

SIG files are device-specific signature files that are required for each device you are using for your application development. These files must be obtained from QUALCOMM and are valid for 90 days. After 90 days, the signature file expires and QUALCOMM must issue a new signature file.

Only one SIG file is required per device. However, because each individual application requires its own corresponding SIG file, you make a copy of the original SIG file and rename it to match the name of the application you have created. Remember that the SIG file must reside in the same folder as the MOD file, the application file for the BREW device.

You must be an authenticated BREW developer to access the signature generation utility on the BREW website.

Note: SIG files are required for testing applications on devices, but they are not required for testing using the BREW Simulator.

Obtain a SIG file

- 1 Go to the BREW website and navigate to the page that contains the link to the QUALCOMM web-based test signature generator.
- 2 Select BREW Testing Generator.
- 3 Enter your BREW developer user name and password.
- 4 In the ESN field, type **0x<ESN Number>** where the ESN number is the number you find under the device battery when you lift it up.
- 5 Click Generate and wait for the Download Signature link to appear.
- 6 Download the signature and save the file; it will be required for all testing on the device.

BAR files

You can add, delete, and move files on a BREW-enabled device with a data cable and a Windows application called AppLoader (QUALCOMM provides AppLoader as part of the BREW Tools Suite). However, you cannot copy files from the BREW device that have the extensions .mod, .mif, or .bar to any other device or storage medium. This limitation is enforced by QUALCOMM to ensure that files purchased through the BREW Delivery System (BDS) cannot be distributed to unauthorized users.

To enable Flash content developers to take advantage of this security feature, the Flash Lite player that runs on BREW devices checks for BAR files when it attempts to reference a file on the local device. This restriction applies to any ActionScript or Shared Libraries function that uses a path to reference a file (for example, `loadMovie()`, `loadSound()`, `loadVariables()`, `XML.load()`, and so on). To use this feature, developers can append the .bar extension to any files they create, and the player can still locate and display them, but no one can copy them from the BREW device onto which they were initially loaded. The Flash Lite player first checks for the existence of the file by using the supplied local path (for example, `loadMovie(example.swf)`). If the file is not found at this location, the player then checks the same path with the .bar extension appended (for example, `loadMovie(example.swf.bar)`). The file is then loaded.

Device file structures for different BREW versions

Devices that support BREW 2.x and BREW 3.x have different file structures. It is important to understand these differences before you use the BREW Publisher wizard to specify your BREW application's file structure.

The file system on BREW 2.x devices is structured as follows:

- All application folders are stored in the root folder, called `brew/`.
- Application files (including SIG files) are stored in the application folder (subfolders are not allowed).
- MIF files are stored in the root folder, `brew/`.
- Shared media, such as images and BAR files are stored in the `brew/shared` folder. You can also store shared media in the folder that contains the main application files.

Example:

`brew/appname/appname.mod`

`brew/appname/appname.sig`

`brew/appname.mif`

`brew/shared/media.xxx`, where xxx is the extension of any resource files, including graphics, video, sound, and BAR files

or

`brew/appname/media.xxx`, where xxx is the extension of any resource files, including graphics, video, sound, and BAR files

The file system on BREW 3.x devices is structured as follows:

- All application directories are stored in the `brew/mod` folder.
- Application files (including SIG files) are stored in the application folder (subfolders are not allowed).
- MIF files are stored in the `brew/mif` folder.
- You can store shared media, such as images and BAR files, in the `brew/shared` folder. You can also store shared media in the folder that contains the main application files.

Example:

`brew/mod/appname/appname.mod`

`brew/mod/appname/appname.sig`

`brew/mif/appname.mif`

`brew/shared/media.xxx`, where xxx is the extension of any resource files, including graphics, video, sound and BAR files

or

`brew/mod/appname/media.xxx`, where xxx is the extension of any resource files, including graphics, video, sound, and BAR files

Workflow for publishing and testing Flash Lite files for BREW

The workflow for publishing and testing Flash Lite files for BREW is as follows:

- In Flash, launch the BREW Publisher wizard and follow the screens to specify the appropriate settings for your BREW application and publish the files.
- In the BREW SDK, use the BREW Simulator to test the application.

Use the BREW Publisher wizard to publish application files

- 1 In Flash, select **File > Publish Settings**.

The Publish Settings dialog box appears.

- 2 On the **Formats** tab, select **Flash (.swf)** if it is not already selected.

You can deselect all other file types, because only the SWF file is needed to generate the BREW applet.

- 3 On the **Flash** tab, from the **Version** list, select a supported version of the player (Flash Player 7 or earlier, or any version of Flash Lite), and then select any of the versions from the **ActionScript** version list (all ActionScript versions are supported).

- 4 From the **Post-Processor** list near the bottom of the **Flash** tab, select **Flash Lite for BREW**.

Note: If the **Post-Processor** list does not appear on the **Flash** tab, check the instructions in “[Setting up your system for BREW](#)” on page 85 to ensure that you correctly installed Flash Lite 2.1 for BREW. The **Post-Processor** list appears only after the software is installed.

- 1 Click **Settings** to the right of the **Post-Processor** list to display the BREW Publisher wizard.

- 2 (Required) Use the input fields on the Identify Applet screen to specify a unique class ID and name for your BREW applet.

The following table lists detailed information about these fields. When you have finished specifying your applet's class ID and name, click Next to continue.

Field name	Field contents
Class ID	This field is prepopulated with a randomly generated value as a default. Replace this value with your certified class ID if you have one. A class ID is a unique 32-bit identification code (8-digit hexadecimal value), which the BREW interface creation mechanism uses. For testing on a local system, you can use the randomly generated default (provided it is unique), although QUALCOMM must issue a unique ID for applications intended for distribution. For more information on obtaining class ID files, see the BREW website.
Applet Name	<p>This field is prepopulated with the root portion of the name of the FLA file that is currently active. For example, if the active FLA file is called BREW fla, the applet name is BREW. The name you enter in this field is displayed in the BREW Application Manager on the target device.</p> <p>Use the following guidelines for Applet names:</p> <ul style="list-style-type: none"> • Applet names must be all lowercase, and must begin with an alphabetic character. • Numbers are allowed, but not as the first character. • The underscore character is the only special character allowed, and it may not be the first character of the name. • The maximum length of the name varies depending on the device. <p>For information, see the device specifications on the BREW website.</p>

- 3 (Optional) Use the Include Applet Information screen to enter information that you want to publish along with your BREW applet.

The information that you enter here is stored in your applet's MIF file. The following table lists detailed information about each field. When you have finished specifying your information, click Next to continue.

Field name	Field contents
Author	Enter the applet author's name (23 characters maximum).
Version	Enter the applet version(23 characters maximum).
Copyright	Enter the copyright information(23 characters maximum).
Domain URL	Enter the fully qualified domain for your applet, ending with a trailing slash (example: http://www.example.com/). The applet's domain controls which information the applet has access to at run time. In earlier versions of the Flash player, SWF files from similar domains (for example, www.exampleapps.com and www.examplegames.com) could communicate freely with each other and with other documents. In Flash Player 7 and later, the domain of the data to be accessed must match the data provider's domain exactly for the domains to communicate. For more information about domains and security, see the Flash documentation's topics on security, cross-domain security, and allowing data access between cross-domain SWF files.
Application Type (Application or Screen Saver)	Specifies whether the BREW applet you are creating is an application or a screen saver. For more information about how screen savers differ from applications, go to the BREW website and navigate to Developer FAQs > BREW Tools > MIF Settings > Screensaver.

- 4 (Optional for testing your application, but mandatory for applications that must pass True BREW Testing) Use the Applet Icons screen to enter or browse for the name and location of one or more icons for your applet.

Supported file types are: JPEG, BMP, PNG, and BCI (BREW Compressed Image). These icons are displayed along with the applet's name in the BREW Application Manager on the target device. When you have finished specifying your icons, click Next to continue.

Field name	Field contents
Small Icon	Image file for small icon (displayed on device). Maximum size: 16 x 16.
Medium Icon	Image file for medium icon (displayed on device). Maximum size: 26 x 26.
Large Icon	Image file for large icon (displayed on distribution server and some devices). Maximum size: 65 x 42.

- 5 Use the Define Output Settings screen to specify the locations in which to publish files for the BREW Simulator and the target device.

These fields are populated with default values derived from your SWF file's location, combined with the applet's name and folder names (simulator and device) that indicate which files to use for testing and which files to upload to the device. You can accept the defaults, or you can specify different output locations. When you have finished, click Next to continue.

Field name	Field contents
For Simulator <ul style="list-style-type: none"> • MIF Folder • Applet Folder 	Contains the files you need to test your applet by using the Simulator tool in the BREW SDK. For further information about the expected file locations and folder structure, see the documentation for the BREW Simulator on the BREW website.
For Device	Contains the files you upload to the BREW device. For more information about how folder structures differ according to the version of BREW supported by the target device, see "Uploading files to BREW devices" on page 97.

- 6 Use the Summary Of Your Selections screen to review the applet settings you selected.

To change any of these settings, use the Back button to return to previous screens and make changes. When your settings are complete, check or deselect the box at the bottom of the screen to specify whether to display the wizard again when you publish, and then click Finish to save your settings.

Field name	Field contents
Summary Of Your Selections	Displays a list of all of your settings for review.
Do Not Show This Wizard On Publish	Controls whether to display the wizard again when you publish the applet files, or trigger the publishing process without redisplaying the wizard. Note that you can always access the wizard regardless of this setting by displaying the Publish Settings dialog box and clicking the Settings button to the right of the Post-Processor list.

- 7 Click OK to save your settings, and then select File > Publish Settings again. The Publish Settings dialog box appears.

Note: This step is required only the first time you select Flash Lite for BREW from the Post-Processor list in a given file. Thereafter, you can publish without clicking OK first.

- 8 Click Publish to publish your files to the locations you specified using the wizard.

Test your application with the BREW Simulator

Before uploading your application to the device, you should test it using the Simulator tool in the BREW SDK. For further information about this tool, see the documentation for the BREW Simulator on the BREW website.

- 1 Navigate to the Simulator folder for the applet that you created by using the BREW Publisher wizard.
- 2 Open the Simulator folder and copy all of its contents to the BREW SDK examples folder.

Note: If you specified the BREW SDK examples folder as the location for the Simulator output files when you used the BREW Publisher wizard to publish the files, the files are already located in the correct folder and do not need to be copied.

- 1 Choose Start > Programs > BREW SDK v<your version> and select the Simulator for BREW to open the application.
- 2 In the Simulator, select File > Load Device, and then select the device pack for your target handset.
- 3 On the Simulator's Properties tab, select your application's folder as the Applet Directory.
- 4 Use the Simulator's tools and features to test your application.

Use device packs

If you are an authenticated BREW developer, you can download device-specific device packs for the devices you are targeting from the Developer Resources area of the BREW website. The device pack for the Simulator may not exactly match your actual device. For example, the Samsung SCH-A950 device pack shows a BACK key, which is labeled as CLR on the actual device. Also, the device pack for the LG VX9800 does not support key input from the alphabetic keys on the keyboard, although the actual device does. Contact QUALCOMM for more information about specific device packs, updates, or issues.

If you are not an authenticated BREW developer, you can use the default device packs that are included with the SDK download. However, because these default device packs have less memory than the device-specific ones, you may need to select the device pack with the most memory (Device Pack 3 at this writing), or increase the memory to approximately 10,000,000 bytes to avoid performance issues and errors.

Increase the memory for a single Simulator session

- 1 Open the Simulator and select the Device Details tab.
- 2 Scroll to the Memory section.
- 3 Increase the value for Heap Size (bytes). Note that this value is called RAM Size in some versions of the Simulator.

Make the memory increase persist across Simulator sessions

- 1 Use a text editor to open the DSD file for the device pack you want to use (devicepack<x>.dsd, where x is the device pack number.)
- 2 Locate the appropriate setting and increase the value of Text:

```
<String Id="24" Name="IDS_DD_HW_RAM_SIZE">  
  <Text>1048576</Text>
```

- 3 Save and close the file.

You may also encounter difficulties using the Simulator if the space allocated to the Examples folder and all of its subfolders is insufficient to allow your application to function properly. You can either delete unnecessary files from these folders manually, or increase the amount of space allocated for these files.

Make the Examples file size increase persist across Simulator sessions

1 Use a text editor to open the configuration file and edit the appropriate value:

- For the Samsung SCH-A950: The configuration file is SCH-A950.qsc. The value to be changed is shown in boldface:

```
FS_LIMITS_PER_MODULE 65535 15204352
```

- For the LG9800: The configuration file is LG-VX9800.dsd. The value to be changed is shown in boldface:

```
<String Id="20" Name="IDS_DD_HW_EFS_SIZE">  
<Text>47000000</Text>  
<Comment>50MB</Comment>  
</String>
```

2 Save and close the file.

Additional resources

For further information about the BREW Simulator, consult the following sources:

Document name and type	Location
<i>Starting with BREW</i> , "About the BREW Tools Suite" (PDF)	Go to the BREW website, and then select Developer Home > Application Developer Documentation.
BREW Simulator (Help)	Open the Simulator and click Help in the menu bar to display the Help.

Uploading files to BREW devices

About uploading Flash Lite applications for BREW

After you create your Flash Lite application and use the Flash Lite for BREW Publisher wizard to generate BREW-compatible files, you upload your application to a supported BREW device and test it.

Ideally you already targeted the device in your initial design process and tested your application by using the Adobe Device Central emulator and the BREW Simulator. However, it is still important to test your application thoroughly on the target device to ensure that it functions as you expect.

Before you can upload your application, you need:

- A device that supports Flash Lite for BREW (at this writing, either a Samsung SCH-A950 or an LG VX9800)
- A serial or USB cable to upload your application from your computer to the device (usually available from the device manufacturer)
- The AppLoader tool (available for download as part of the BREW Tools Suite to authenticated developers from the BREW website)
- Application files in the required file formats and structure (created by using the Flash Lite for BREW Publisher wizard)

These are the required application files:

- *appname*.SIG (available for download to authenticated developers from the QUALCOMM web-based test signature generator)
- *appname*.MOD (generated by the Flash Lite Publisher for BREW wizard)

- *appname.MIF* (generated by the Flash Lite Publisher for BREW wizard)
- All sound, video, image, and BAR files that your application requires

For complete information about required file formats and structure for your target device, see “[Device file structures for different BREW versions](#)” on page 92

Workflow for uploading applications

This workflow assumes that you have already installed the appropriate USB drivers for your device on your computer. For more information, see “[Setting up your system for BREW](#)” on page 85.

To upload and test Flash Lite applications on BREW devices, complete the following tasks:

- 1 Connect the handset to the computer with a cable.
- 2 Open AppLoader on the computer:
 - a Copy application files to the appropriate locations for the device. For information, see “[Device file structures for different BREW versions](#)” on page 92.
 - b If necessary, copy Flash Lite Extension and Player files to the device (first time only).
 - c Restart the device.
- 3 Test the application on the device.

The rest of this section contains more detailed information about each of these tasks.

Uploading Flash Lite Extension files to the device (first use only)

Before you can test your Flash Lite for BREW applications on a given device, copy the Flash Lite Extension onto the device. This needs to be done only once, when you first acquire the device for testing. End users of your application will receive these files as part of the download when they download your completed application from the BDS or ADS, but developers must copy these files manually. The files are listed below. Follow the instructions in the rest of this section to upload the files to your device using the AppLoader tool. Note that the locations to which you copy these files are different for BREW 2.x and 3.x devices. See “[Device file structures for different BREW versions](#)” on page 92 for more information.

Extension file	Source
flashlite_2_1.sig	Generated by developers using a tool available from the BREW website.
flashlite_2_1.mif	Available for download from the BREW website.
flashlite_2_1.mod	Available for download from the BREW website.
brewsaplayer.sig	Generated by developers using a tool available from the BREW website.
brewsaplayer.mif	Available for download from the BREW website.
brewsaplayer.mod	Available for download from the BREW website.

AppLoader hints and best practices

The following is a list of hints for using the AppLoader tool:

- When you create a folder on the device, do not use the backward slash (\).
- Do not run any applications on the device while you are copying files.
- Restart the device after you use AppLoader to make any changes.

- Avoid overwriting files in AppLoader; instead, delete the old versions of files on the device and replace them with new copies of the files.
- Do not upload files or folders whose names consist of numerals only. You will be unable to delete these files and folders from some devices.
- Do not begin a file or folder name with the characters “shared.” “Shared” is a reserved word in BREW, and you will be prevented from uploading files or folders that begin with those characters.

Upload applications to a BREW 2.x device

The following instructions explain how to upload a Flash Lite application for BREW to a Samsung SCH-A950, which includes the BREW version 2.x platform.

This procedure assumes that you have already installed the appropriate USB drivers for your device on your computer. For more information, see [“Setting up your system for BREW”](#) on page 85.

This procedure also assumes that you have already uploaded the required Flash Lite Extension and Flash Lite player files to the device. This needs to be done only once, when you first begin using the device. For more information, see [“Uploading Flash Lite Extension files to the device \(first use only\)”](#) on page 98.

- 1 Use the data cable that the handset manufacturer provides to attach the handset to a COM port on the computer on which your BREW application files are stored.
- 2 Navigate to the folder that contains your application files and verify that the directory structure is correct for your target device.

For complete information about the structure for BREW 2.x devices, see [“Device file structures for different BREW versions”](#) on page 92.

- 3 Select Start > Programs > BREW Tools Suite <latest version> > BREW Apploader to start the AppLoader tool. The AppLoader tool displays a connection window.
- 4 Select the number of the COM port your device is connected to (if you are unsure what port to select, see the following note), select 2.x as the BREW version for your device, and then click OK to connect to the device.

Note: You can find the COM port your device is connected to in the Windows Device Manager by looking at the properties of the port that corresponds to the installed device. If you always connect the same device model to the same COM port, this number does not change. Because it changes when you use a different device model, you should always check the COM port number when you connect a new device model.

After you are connected, the AppLoader tool displays the device’s BREW file system.

- 1 To upload your application files (MOD, SIG, and resource files) to the handset, do one of the following:
 - Select File > Directory > New Directory to create a folder with the same name as your application, and then copy your application files to it. See [“Device file structures for different BREW versions”](#) on page 92 for details.
 - Drag your application folder from the Windows Explorer to the AppLoader window.
- 2 Restart the device.

The device must be restarted before you can run your application. To restart the device, do one of the following:

- Select Device > Reset from the BREW AppLoader menu.
- Press and hold the End key on the handset.

- 3 Navigate to Get It Now > Get Going and select the name of your application to start it in the BREW environment.

Upload applications to a BREW 3.x device

The instructions that follow explain how to upload a Flash Lite application for BREW to an LG VX9800, which includes the BREW version 3.x platform.

This procedure assumes that you have already installed the appropriate USB drivers for your device on your computer. For more information, see “[Setting up your system for BREW](#)” on page 85.

This procedure also assumes that you have already uploaded the required Flash Lite Extension and Flash Lite player files to the device. This needs to be done only once, when you first begin using the device. For more information, see “[Uploading Flash Lite Extension files to the device \(first use only\)](#)” on page 98.

- 1 Use the cable that the handset manufacturer provides to attach the handset to a COM port on the computer on which your BREW application files are stored.
- 2 Navigate to the folder that contains your application files and verify that the directory structure is correct for your target device.

For complete information about the structure for BREW 3.x devices, see “[Device file structures for different BREW versions](#)” on page 92.

- 3 Select Start > Programs > BREW Tools Suite <latest version> > BREW Apploader to start the AppLoader tool.
- 4 The AppLoader tool displays a connection window.

Select the number of the COM port your device is connected to, select 3.x as the BREW version for your device, and then click OK to connect to the device.

Note: You can find the COM port that your device is connected to in the Windows Device Manager by looking at the properties of the port that corresponds to the installed device. If you always connect the same device model to the same COM port, this number does not change. Because it changes when you use a different device model, you should always check the COM port number when you connect a new device model.

After you are connected, the AppLoader tool displays the device’s BREW file system.

- 1 To upload your application files (MOD, SIG and resource files) to the handset, do one of the following:
 - Select File > New > Directory > New Directory to create a directory with the same name as your application, and then copy your application files to it. See “[Device file structures for different BREW versions](#)” on page 92 for details.
 - Drag your application folder from the Windows Explorer to the AppLoader window.
- 2 To restart the device, do one of the following:
 - Select Device > Reset from the BREW AppLoader menu.
 - Press and hold the End key on the handset.
- 3 Navigate to Get It Now > Get Going and select the name of your application to start it in the BREW environment.

On some devices, your application will not be displayed in the standard Get Going menu. To display your application on these devices, follow these steps:

- a On the Verizon homescreen, press Select/OK to access the menu.
- b Press the zero key (0) to access the service menu.
- c Enter the default password, which is six zeros (000000).
- d Press the nine key (9) to display Get It Now.
- e Press the g key to access the screen on which Flash Lite applications are displayed.

Testing applications on the device

Before you can start and test your application on the device, you must have a test signature from QUALCOMM. Authenticated developers can use a web-based tool called Testsig on the BREW Developer extranet to generate test signatures. For specific information on how to generate and use your Testsig, see the BREW website. You can find extensive information on how to test your BREW applets to ensure that they ultimately comply with TBT (True BREW Testing) standards on the BREW Developer extranet.

Additional resources

For further information about the AppLoader tool, consult the following sources:

Document name and type	Location
<i>Starting with BREW</i> , "About the BREW Tools Suite" (PDF)	Go to the BREW website and then select Developer Home > Application Developer Documentation
BREW_Apploader.chm (Help)	C:\Program Files\BREW Tools Suite <latest version>\BREWApploader (default)
BREW Testing generator	Go to the BREW website and navigate to the SDK download page

Chapter 9: Testing Flash Lite content

Adobe Flash CS4 Professional includes an Adobe Flash Lite emulator available on Adobe Device Central CS4 that lets you test your application in the authoring tool as it will appear and function on an actual device. When you're satisfied with the application running in the emulator, you can test it on an actual device.

Testing overview

Flash Lite testing features

The Flash Lite testing features in Flash CS4 Professional are part of Adobe Device Central, which includes both an extensive database of device profiles and a device emulator. Device Central also works with many other Adobe products, such as Adobe Creative Suite® and Adobe Dreamweaver®.

The Adobe Device Central emulator lets you preview your Flash Lite content within the Flash authoring tool. The emulator is configured to match the behavior and appearance of the target device. You use the Device Central interface to select and manage your target devices, as well as to specify your application's target Flash Lite content type, such as ring tone, browser, or stand-alone application. In Device Central, each combination of test device and Flash Lite content type defines a device configuration that specifies what features are available to your application, such as supported audio formats, ability to make network connections, and other features. For more information about the Flash Lite content types, see [“Supported content types”](#) on page 104.

See Device Central Help for more information, including details about interacting with the emulator.

Testing features not supported by the emulator

The Adobe Device Central emulator does not support all the features available in the standard (desktop) test window. The following is a list of features available in the desktop Flash test window that are not available in the Adobe Device Central emulator:

- The List Variables (Debug > List Variables) and List Objects (Debug > List Objects) features
- The Bandwidth Profiler, and Streaming and Frame by Frame graphing features
- The View > Simulate Download menu command
- The ActionScript debugger
- The View > Show Redraw Regions menu command
- The Controller toolbar (Window > Toolbars > Controller)

Testing inline text in Flash Lite 2.1 and later

You cannot currently test the inline text functionality in the Adobe Device Central emulator; you must test this feature on a device. When testing in the emulator, you must edit the contents of input text fields using a modal dialog box that appears over the Flash Lite content (that is, the emulator functions the same way it did for Flash Lite 1.x and Flash Lite 2.0). For details on how the modal dialog box works, see [“Using input text fields”](#) on page 51.

Using the emulator

Invoking Adobe Device Central

To start the emulator, choose Control > Test Movie in the Flash authoring tool, the same way you preview your Flash desktop content. (However, Adobe Device Central has a different appearance and functionality than the test window for Flash desktop content.) After you choose Control > Test Movie or press Ctrl+Enter to start Device Central, a progress bar appears to indicate that Flash is exporting your SWF to Device Central. Once the export is complete, Device Central launches with the focus on the emulator, and loads your SWF.

In Device Central, the Device Sets list shows all target devices that Flash saved with your application. By default, the first device in the set is selected for emulation. The content type selected is the content type that was saved with the application when you created it (for details about the Flash Lite content types, see “Supported content types” on page 104).

You can test the application with a different content type and different devices. Changing the content type and adding or removing devices from the device set automatically changes the device settings in Flash.

To test how the content appears on a different device, double-click another device in either the top or bottom list. A spinning icon appears next to the device you are testing, and the emulator changes to display your application running in the device you selected.

Set emulator debug options

The Adobe Device Central emulator can send debugging messages to the Flash Output panel while content is running; the emulator also displays a pop-up version of the Output panel showing the same messages.

The emulator reports the following types of information:

Trace messages that are generated by a `trace()` function call within your Flash Lite application. For more information about using `trace()`, see `trace()` in *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Information messages that contain general information about the selected test device, SWF file size, and other information. These messages appear in the emulator’s Alert panel.

Warning messages that contain information about problems with your Flash Lite content that might affect playback.

You can filter the type of information that the emulator generates as follows.

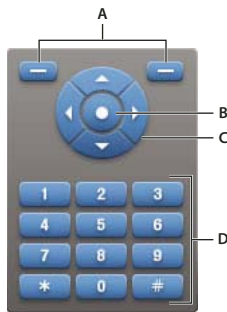
- 1 Select Control > Test Movie. Flash exports your application to Adobe Device Central and displays it in the emulator.
- 2 Select View > Flash Output > Show In Device Central.
 - Select or deselect the Trace option.
 - Select or deselect the Information option.
 - Select or deselect the Warnings option.

Interacting with the emulator

You can use your computer mouse or use keyboard shortcuts to interact with the emulator’s keypad. You can interact with the following keys on the emulator’s keypad:

- Number keys (0 to 9), and the asterisk (*) and pound (#) keys
- Navigation keypad (left, right, down, up, and select)

- Left and Right soft keys



A. Soft keys B. Select key C. Navigational keypad D. Other supported keys

You can use your mouse to click the emulator's keypad directly, or you can use the following equivalent keyboard shortcuts:

- The arrow keys on your keyboard (left, right, up, down) map to the corresponding navigation keys on the emulator's navigation keypad.
- The Enter or Return key corresponds to the emulator's select key.
- The Page Up and Page Down keys correspond to the emulator's Left and Right soft keys, respectively.
- The number keys on your keyboard map to the corresponding number keys on the emulator keypad.

For additional details about interacting with the emulator to test your application, see the Adobe Device Central online help.

Supported content types

Flash Lite is installed on a variety of devices. Each installation supports one or more application modes, or *content types*. For example, some devices use Flash Lite to enable screen savers or animated ring tones. Other devices use Flash Lite to render Flash content that is embedded in mobile web pages.

The following table lists and describes all of the Flash Lite content types available as of this writing. For additional and up-to-date information about Flash Lite content-type availability, see the Flash Enabled Mobile Device page at www.adobe.com/go/mobile_supported_devices/.

Note: As of this writing, Flash Lite 2.0 supports the Standalone Player content type only.

Flash Lite content type	Description	Availability
Address Book	Uses Flash Lite to let users associate a SWF file with an entry in their device's address book application.	DoCoMo and VodafoneKK (Japan only)
Alarm	Uses Flash Lite to let the user select a SWF file to play for the device's alarm.	KDDI and VodafoneKK (Japan only)
Browser	Uses Flash Lite to render Flash content embedded in mobile web pages and viewed in the device's web browser.	DoCoMo, KDDI, and VodafoneKK (Japan only)
Calling History	Uses Flash Lite to display an image or animation associated with each entry in the user's address book, along with the name and phone number.	KDDI (Casio phones only)

Flash Lite content type	Description	Availability
Calling Screen	Uses Flash Lite to display an animation when the user receives a call or makes a call.	DoCoMo and KDDI (Japan only)
Chaku Flash	Uses Flash Lite to let the user select a SWF file to play as the ring tone for incoming calls.	KDDI (Japan only)
Data Box	Uses Flash Lite to render Flash content in the device's Data Box application, which lets the user manage and preview multimedia files on the device.	DoCoMo, KDDI, and VodafoneKK (Japan only)
Data Folder	Uses Flash Lite to render Flash content in the device's Data Folder application, which lets the user manage and preview multimedia files on the device.	KDDI (Japan only)
Icon Menu	Uses Flash Lite to let the user select custom icon menus for the device's launcher application (used by the UILauncher content type).	KDDI (Casio phones only)
Image Viewer	Use the Image Viewer application that lets the user manage and preview multimedia files on the device, including SWF files.	DoCoMo (Japan only)
Incoming Call	Uses Flash Lite to display an animation when the user receives a call.	DoCoMo, KDDI, and VodafoneKK (Japan only)
Mailer	Uses Flash Lite to display an animation when the user sends or receives an e-mail message.	VodafoneKK (Japan only)
Multimedia	Uses Flash Lite to preview SWF files (as well as other multimedia formats).	KDDI (Japan only)
My Picture	Uses the My Picture application that lets the user manage and preview SWF files on the device, as well as other image formats.	DoCoMo (Japan only)
OpenEMIRO	Displays Flash Lite content when the device is returning from Standby mode. This is similar to the Wake Up Screen content type on other devices.	KDDI (Casio devices only)
Screen Saver	Uses Flash Lite to display the device's screen saver.	KDDI and VodafoneKK (Japan only)
SMIL Player	Uses Flash Lite to preview SWF files (as well as other multimedia formats).	KDDI (Japan only)
Standalone Player	Makes Flash Lite available as a stand-alone application so that the user can start and view arbitrary SWF files that reside on the device or that the user receives in the messaging inbox.	Available globally for select Symbian Series 60 and UIQ devices
Standby Screen	Uses Flash Lite to display the device's Standby Screen (or wallpaper screen).	DoCoMo and KDDI (Japan only)

Flash Lite content type	Description	Availability
Sub LCD	Uses Flash Lite to display content on the external or secondary screen available on some flip phones.	KDDI (Japan only)
UILauncher	Uses Flash Lite for the device's application launcher.	Uses Flash Lite to display the device's launcher application (that is, the application that lets the user start other applications).
Wake Up Screen	Uses Flash Lite to display an animation as the phone is starting.	DoCoMo (Japan only)

Flash Lite specific information in the emulator

The emulator includes panels that supply information specific to your Flash Lite application. The panels appear along the right side of the window; you can collapse and expand them as you do in Flash.

For details about these panels and how to use them, see the Adobe Device Central online help.

About screen size and available Stage size

Each combination of target device and Flash Lite content type determines, among other things, the available screen area that a Flash Lite application can occupy. The available Stage area can be equal to, or less than, the device's full screen size.

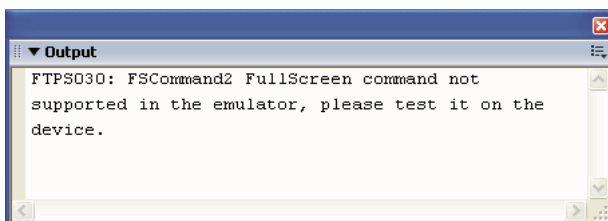
For example, the Stage area that is available to a SWF file running in full-screen mode in the stand-alone player on a Nokia Series 60 device is equal to the device's full screen size (176 x 208 pixels). On other devices (such as those available in Japan), the Stage area that is available to a SWF file running in one of the specialized content types (such as Address Book or Screensaver) might be less than the device's total screen size. For example, the Fujitsu 700i has a screen size of 240 x 320 pixels; however, a SWF file running in the device's Address Book application has 96 x 72 pixels of available Stage area.

If a SWF file's actual dimensions are different from the available Stage dimensions, the Flash Lite player scales the content (proportionately) to fit within the available Stage area. When you test your content in the emulator, the emulator also warns if your application's Stage size is different from the available Stage area.

To avoid any undesired scaling issues, Adobe recommends that your Flash document's Stage dimensions match the available Stage area for the selected test device and content type.

Flash Lite features not supported by the emulator

The emulator doesn't support all the ActionScript commands and player features that are available to Flash Lite applications running on an actual device. For example, the emulator doesn't support the ability to initiate phone calls or Short Message Service (SMS) messages. If you attempt to use a command or feature that isn't supported by the emulator, the emulator generates a message in the Output panel, as the following image shows:



You must test your SWF file on an actual device to confirm that those features are functioning as expected.

The Adobe Device Central emulator does not support the following `fscommand()` and `fscommand2()` commands:

- `FullScreen`
- `GetFreePlayerMemory`
- `GetTotalPlayerMemory`
- `Launch`
- `Quit`
- `StartVibrate`
- `GetNetworkConnectStatus`
- `GetNetworkRequestStatus`
- `GetNetworkStatus`

Playing a device video in the emulator

The Flash Lite player uses the device's default video handler application to play video content in your SWF file, rather than decoding the video natively. This practice lets Flash Lite developers use any video format that the target device supports, such as 3GPP, MPEG, or AVI. For more information about using video in Flash Lite, see [“Working with video”](#) on page 71.

The Adobe Device Central emulator uses QuickTime Player to render device video when testing in the Flash authoring tool. The latest version of QuickTime Player (version 7 as of this writing) supports playback of several different device video formats, including 3GPP and others. However, by default, QuickTime might not support some video formats during playback that an actual device supports, and therefore those formats will not play in the Adobe Device Central emulator. For this reason, it's important to always test your content on an actual device.

If your device video does not play in the QuickTime Player, by default, try the following:

- Upgrade to the latest version of QuickTime Player.
- If available, install a third-party video codec (short for compressor-decompressor) that supports the video format you're using.

Chapter 10: Introduction to Flash Lite 2.x and 3.x ActionScript

Macromedia Flash Lite 1.0 and Macromedia Flash Lite 1.1 software from Adobe, the first versions of Flash Lite, are based on Macromedia Flash Player 4 software from Adobe. Macromedia Flash Lite 2.0 and Macromedia Flash Lite 2.1 software from Adobe, along with Adobe Flash Lite 3.0 and 3.1, are based on Macromedia Flash Player 7 from Adobe, but differ from it in the following respects:

- Flash Lite supports some features only partially.
- Flash Lite adds some features specifically for mobile devices.

This document describes the differences between the Adobe ActionScript supported for Flash Lite 2.0, 2.1, 3.0 and 3.1 (which are referenced collectively as 2.x and 3.x respectively) and the ActionScript that was supported for Flash Player 7.

Supported, partially supported, and unsupported ActionScript classes and language elements

Accessibility

The following table shows which versions of Flash Lite support the Accessibility class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
isActive() : Boolean	Yes	No	No	No
updateProperties() : Void	Yes	No	No	No

arguments

The following table shows which versions of Flash Lite support the arguments class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
callee:Object	Yes	Yes	Yes	Yes
caller:Object	Yes	Yes	Yes	Yes
length:Number	Yes	Yes	Yes	Yes

Array

The following table shows which versions of Flash Lite support the Array class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
CASEINSENSITIVE:Number	Yes	Yes	Yes	Yes
concat([value:Object]) : Array	Yes	Yes	Yes	Yes
DESCENDING:Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
join([delimiter:String]) : String	Yes	Yes	Yes	Yes
length:Number	Yes	Yes	Yes	Yes
NUMERIC:Number	Yes	Yes	Yes	Yes
pop() : Object	Yes	Yes	Yes	Yes
push(value:Object) : Number	Yes	Yes	Yes	Yes
RETURNINDEXEDARRAY:Number	Yes	Yes	Yes	Yes
reverse() : Void	Yes	Yes	Yes	Yes
shift() : Object	Yes	Yes	Yes	Yes
slice([startIndex:Number], [endIndex:Number]) : Array	Yes	Yes	Yes	Yes
sort([compareFunction:Object], [options:Number]) : Array	Yes	Yes	Yes	Yes
sortOn(fieldName:Object, [options:Object]) : Array	Yes	Yes	Yes	Yes
splice(startIndex:Number, [deleteCount:Number], [value:Object]) : Array	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
UNIQUESORT:Number	Yes	Yes	Yes	Yes
unshift(value:Object) : Number	Yes	Yes	Yes	Yes

AsBroadcaster

The following table shows which versions of Flash Lite support the AsBroadcaster class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_listeners:Array [read-only]	Yes	No	Yes	Yes
addListener(listenerObj:Object) : Boolean	Yes	No	Yes	Yes
broadcastMessage(eventName:String) : Void	Yes	No	Yes	Yes
initialize(obj:Object) : Void	Yes	No	Yes	Yes
removeListener(listenerObj:Object) : Boolean	Yes	No	Yes	Yes

BevelFilter

The following table shows which versions of Flash Lite support the BevelFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
angle:Number	Yes	No	No	No
BevelFilter([distance:Number], [angle:Number], [highlightColor:Number], [highlightAlpha:Number], [shadowColor:Number], [shadowAlpha:Number], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
clone() : BevelFilter	Yes	No	No	No
distance:Number	Yes	No	No	No
highlightAlpha:Number	Yes	No	No	No
highlightColor:Number	Yes	No	No	No
knockout:Boolean	Yes	No	No	No
quality:Number	Yes	No	No	No
shadowAlpha:Number	Yes	No	No	No
shadowColor:Number	Yes	No	No	No
strength:Number	Yes	No	No	No
type:String	Yes	No	No	No

BitmapData

The following table shows which versions of Flash Lite support the BitmapData class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
applyFilter(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, filter:BitmapFilter) : Number	Yes	No	No	No
BitmapData(width:Number, height:Number, [transparent:Boolean], [fillColor:Number])	Yes	No	No	Yes
clone() : BitmapData	Yes	No	No	Yes
colorTransform(rect:Rectangle, colorTransform:ColorTransform) : Void	Yes	No	No	Yes
compare(otherBitmapData:BitmapData) : Object	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
copyChannel(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, sourceChannel:Number, destChannel:Number) : Void	Yes	No	No	Yes
copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [alphaBitmap:BitmapData], [alphaPoint:Point], [mergeAlpha:Boolean]) : Void	Yes	No	No	Partial
dispose() : Void	Yes	No	No	Yes
draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform], [blendMode:Object], [clipRect:Rectangle], [smooth:Boolean]) : Void	Yes	No	No	Partial
fillRect(rect:Rectangle, color:Number) : Void	Yes	No	No	Yes
floodFill(x:Number, y:Number, color:Number) : Void	Yes	No	No	Yes
generateFilterRect(sourceRect:Rectangle, filter:BitmapFilter) : Rectangle	Yes	No	No	No
getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]) : Rectangle	Yes	No	No	Yes
getPixel(x:Number, y:Number) : Number	Yes	No	No	Yes
getPixel32(x:Number, y:Number) : Number	Yes	No	No	Yes
height:Number [read-only]	Yes	No	No	Yes
hitTest(firstPoint:Point, firstAlphaThreshold:Number, secondObject:Object, [secondBitmapPoint:Point], [secondAlphaThreshold:Number]) : Boolean	Yes	No	No	Yes
loadBitmap(id:String) : BitmapData	Yes	No	No	Yes
merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number) : Void	Yes	No	No	Yes
noise(randomSeed:Number, [low:Number], [high:Number], [channelOptions:Number], [grayScale:Boolean]) : Void	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
paletteMap(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [redArray:Array], [greenArray:Array], [blueArray:Array], [alphaArray:Array]) : Void	Yes	No	No	No
perlinNoise(baseX:Number, baseY:Number, numOctaves:Number, randomSeed:Number, stitch:Boolean, fractalNoise:Boolean, [channelOptions:Number], [grayScale:Boolean], [offsets:Object]) : Void	Yes	No	No	No
pixelDissolve(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [randomSeed:Number], [numberOfPixels:Number], [fillColor:Number]) : Number	Yes	No	No	No
rectangle:Rectangle [read-only]	Yes	No	No	Yes
scroll(x:Number, y:Number) : Void	Yes	No	No	No
setPixel(x:Number, y:Number, color:Number) : Void	Yes	No	No	Yes
setPixel32(x:Number, y:Number, color:Number) : Void	Yes	No	No	Yes
threshold(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, operation:String, threshold:Number, [color:Number], [mask:Number], [copySource:Boolean]) : Number	Yes	No	No	No
transparent:Boolean [read-only]	Yes	No	No	Yes
width:Number [read-only]	Yes	No	No	Yes

BitmapFilter

The following table shows which versions of Flash Lite support the BitmapFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
clone() : BitmapFilter	Yes	No	No	No

BlurFilter

The following table shows which versions of Flash Lite support the BlurFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
BlurFilter([blurX:Number], [blurY:Number], [quality:Number])	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
clone() : BlurFilter	Yes	No	No	No
quality:Number	Yes	No	No	No

Boolean

The following table shows which versions of Flash Lite support the Boolean class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
Boolean([value:Object])	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
valueOf() : Boolean	Yes	Yes	Yes	Yes

Button

The following table shows which versions of Flash Lite support the Button class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_alpha:Number	Yes	Yes	Yes	Yes
_focusrect:Boolean	Yes	Yes	Yes	Yes
_height:Number	Yes	Yes	Yes	Yes
_highquality:Number	Deprecated	Deprecated	Deprecated	Deprecated
_name:String	Yes	Yes	Yes	Yes
_parent:MovieClip	Yes	Yes	Yes	Yes
_quality:String	Yes	Yes	Yes	Yes
_rotation:Number	Yes	Yes	Yes	Yes
_soundbuftime:Number	Yes	Yes	Yes	Yes
_target:String [read-only]	Yes	Yes	Yes	Yes
_url:String [read-only]	Yes	Yes	Yes	Yes
_visible:Boolean	Yes	Yes	Yes	Yes
_width:Number	Yes	Yes	Yes	Yes
_x:Number	Yes	Yes	Yes	Yes
_xmouse:Number [read-only]	Yes	Yes	Yes	Yes
_xscale:Number	Yes	Yes	Yes	Yes
_y:Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_ymouse:Number [read-only]	Yes	Yes	Yes	Yes
_yscale:Number	Yes	Yes	Yes	Yes
blendMode:Object	Yes	No	No	No
cacheAsBitmap:Boolean	Yes	No	No	No
enabled:Boolean	Yes	Yes	Yes	Yes
filters:Array	Yes	No	No	No
getDepth() : Number	Yes	Yes	Yes	Yes
menu:ContextMenu	Yes	No	No	No
onDragOut = function() {}	Yes	Yes	Yes	Yes
onDragOver = function() {}	Yes	Yes	Yes	Yes
onKeyDown = function() {}	Yes	Yes	Yes	Yes
onKeyUp = function() {}	Yes	Yes	Yes	Yes
onKillFocus = function(newFocus:Object) {}	Yes	Yes	Yes	Yes
onPress = function() {}	Yes	Yes	Yes	Yes
onRelease = function() {}	Yes	Yes	Yes	Yes
onReleaseOutside = function() {}	Yes	Yes	Yes	Yes
onRollOut = function() {}	Yes	Yes	Yes	Yes
onRollOver = function() {}	Yes	Yes	Yes	Yes
onSetFocus = function(oldFocus:Object) {}	Yes	Yes	Yes	Yes
scale9Grid:Rectangle	Yes	No	No	No
tabEnabled:Boolean	Yes	Yes	Yes	Yes
tabIndex:Number	Yes	Yes	Yes	Yes
trackAsMenu:Boolean	Yes	Yes	Yes	Yes
useHandCursor:Boolean	Yes	No	No	No

Camera

The following table shows which versions of Flash Lite support the Camera class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
activityLevel:Number [read-only]	Yes	No	No	No
bandwidth:Number [read-only]	Yes	No	No	No
currentFps:Number [read-only]	Yes	No	No	No
fps:Number [read-only]	Yes	No	No	No
get([index:Number]) : Camera	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
height:Number [read-only]	Yes	No	No	No
index:Number [read-only]	Yes	No	No	No
motionLevel:Number [read-only]	Yes	No	No	No
motionTimeOut:Number [read-only]	Yes	No	No	No
muted:Boolean [read-only]	Yes	No	No	No
name:String [read-only]	Yes	No	No	No
names:Array [read-only]	Yes	No	No	No
onActivity = function(active:Boolean) {}	Yes	No	No	No
onStatus = function(infoObject:Object) {}	Yes	No	No	No
quality:Number [read-only]	Yes	No	No	No
setMode([width:Number], [height:Number], [fps:Number], [favorArea:Boolean]) : Void	Yes	No	No	No
setMotionLevel([motionLevel:Number], [timeOut:Number]) : Void	Yes	No	No	No
setQuality([bandwidth:Number], [quality:Number]) : Void	Yes	No	No	No
width:Number [read-only]	Yes	No	No	No

capabilities

The following table shows which versions of Flash Lite support the capabilities class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
audioMIMETypes:Array [read-only]	No	Yes	Yes	Yes
avHardwareDisable:Boolean [read-only]	Yes	Yes	Yes	Yes
has4WayKeyAS:Boolean [read-only]	No	Yes	Yes	Yes
hasAccessibility:Boolean [read-only]	Yes	Yes	Yes	Yes
hasAudio:Boolean [read-only]	Yes	Yes	Yes	Yes
hasAudioEncoder:Boolean [read-only]	Yes	Yes	Yes	Yes
hasCMIDI:Boolean [read-only]	No	Yes	Yes	Yes
hasCompoundSound:Boolean [read-only]	No	Yes	Yes	Yes
hasDataLoading:Boolean [read-only]	No	Yes	Yes	Yes
hasEmail:Boolean [read-only]	No	Yes	Yes	Yes
hasEmbeddedVideo:Boolean [read-only]	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
hasIME:Boolean [read-only]	Yes	No	No	No
hasMappableSoftKeys:Boolean	No	Yes	Yes	Yes
hasMFI:Boolean [read-only]	No	Yes	Yes	Yes
hasMIDI:Boolean [read-only]	No	Yes	Yes	Yes
hasMMS:Boolean [read-only]	No	Yes	Yes	Yes
hasMouse:Boolean [read-only]	No	Yes	Yes	Yes
hasMP3:Boolean [read-only]	Yes	Yes	Yes	Yes
hasPrinting:Boolean [read-only]	Yes	Yes	Yes	Yes
hasQWERTYKeyboard:Boolean [read-only]	No	Yes	Yes	Yes
hasScreenBroadcast:Boolean [read-only]	Yes	Yes	Yes	Yes
hasScreenPlayback:Boolean [read-only]	Yes	Yes	Yes	Yes
hasSharedObjects:Boolean [read-only]	No	Yes	Yes	Yes
hasSMAF:Boolean [read-only]	No	Yes	Yes	Yes
hasSMS:Number [read-only]	No	Yes	Yes	Yes
hasStreamingAudio:Boolean [read-only]	Yes	Yes	Yes	Yes
hasStreamingVideo:Boolean [read-only]	Yes	Yes	Yes	Yes
hasStylus:Boolean [read-only]	No	Yes	Yes	Yes
hasVideoEncoder:Boolean [read-only]	Yes	Yes	Yes	Yes
isDebugger:Boolean [read-only]	Yes	Yes	Yes	Yes
language:String [read-only]	Yes	No	No	No
localFileReadDisable:Boolean [read-only]	Yes	Yes	Yes	Yes
manufacturer:String [read-only]	Yes	No	No	No
MIMETypes:Array [read-only]	No	Yes	Yes	Yes
os:String [read-only]	Yes	No	No	No
pixelAspectRatio:Number [read-only]	Yes	No	No	No
playerType:String [read-only]	Yes	No	No	No
screenColor:String [read-only]	Yes	No	No	No
screenDPI:Number [read-only]	Yes	No	No	No
screenOrientation:String [read-only]	No	Yes	Yes	Yes
screenResolutionX:Number [read-only]	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
screenResolutionY:Number [read-only]	Yes	Yes	Yes	Yes
serverString:String [read-only]	Yes	No	No	No
softKeyCount:Number [read-only]	No	Yes	Yes	Yes
version:String [read-only]	Yes	Yes	Yes	Yes
videoMIMEtypes:Array [read-only]	No	Yes	Yes	Yes

Color

The following table shows which versions of Flash Lite support the Color class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getTransform() : Object	Deprecated	Yes	Deprecated	Deprecated
setRGB(offset:Number) : Void	Deprecated	Yes	Deprecated	Deprecated
setTransform(transformObject:Object) : Void	Deprecated	Yes	Deprecated	Deprecated

ColorMatrixFilter

The following table shows which versions of Flash Lite support the ColorMatrixFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
clone() : ColorMatrixFilter	Yes	No	No	No
ColorMatrixFilter(matrix:Array)	Yes	No	No	No
matrix:Array	Yes	No	No	No

ColorTransform

The following table shows which versions of Flash Lite support the ColorTransform class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alphaMultiplier:Number	Yes	No	No	Yes
alphaOffset:Number	Yes	No	No	Yes
blueMultiplier:Number	Yes	No	No	Yes
blueOffset:Number	Yes	No	No	Yes
ColorTransform([redMultiplier:Number], [greenMultiplier:Number], [blueMultiplier:Number], [alphaMultiplier:Number], [redOffset:Number], [greenOffset:Number], [blueOffset:Number], [alphaOffset:Number])	Yes	No	No	Yes
concat(second:ColorTransform) : Void	Yes	No	No	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
greenMultiplier:Number	Yes	No	No	Yes
greenOffset:Number	Yes	No	No	Yes
redMultiplier:Number	Yes	No	No	Yes
redOffset:Number	Yes	No	No	Yes
rgb:Number	Yes	No	No	Yes
toString() : String	Yes	No	No	Yes

ContextMenu

The following table shows which versions of Flash Lite support the ContextMenu class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
builtInItems:Object	Yes	No	No	No
ContextMenu([callbackFunction:Function])	Yes	No	No	No
customItems:Array	Yes	No	No	No
hideBuiltInItems() : Void	Yes	No	No	No
onSelect = function(item:Object, item_menu:Object) {}	Yes	No	No	No

ContextMenuitem

The following table shows which versions of Flash Lite support the ContextMenuItem class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
caption:String	Yes	No	No	No
ContextMenuItem(caption:String, callbackFunction:Function, [separatorBefore:Boolean], [enabled:Boolean], [visible:Boolean])	Yes	No	No	No
copy() : ContextMenuItem	Yes	No	No	No
enabled:Boolean	Yes	No	No	No
onSelect = function(obj:Object, menuItem:Object) {}	Yes	No	No	No
separatorBefore:Boolean	Yes	No	No	No
visible:Boolean	Yes	No	No	No

ConvolutionFilter

The following table shows which versions of Flash Lite support the ConvolutionFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alpha:Number	Yes	No	No	No
bias:Number	Yes	No	No	No
clamp:Boolean	Yes	No	No	No
clone() : ConvolutionFilter	Yes	No	No	No
color:Number	Yes	No	No	No
ConvolutionFilter(matrixX:Number, matrixY:Number, matrix:Array, [divisor:Number], [bias:Number], [preserveAlpha:Boolean], [clamp:Boolean], [color:Number], [alpha:Number])	Yes	No	No	No
divisor:Number	Yes	No	No	No
matrix:Array	Yes	No	No	No
matrixX:Number	Yes	No	No	No
matrixY:Number	Yes	No	No	No
preserveAlpha:Boolean	Yes	No	No	No

CustomActions

The following table shows which versions of Flash Lite support the CustomActions class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
get(name:String) : String	Yes	No	No	No
install(name:String, data:String) : Boolean	Yes	No	No	No
list() : Array	Yes	No	No	No
uninstall(name:String) : Boolean	Yes	No	No	No

Date

The following table shows which versions of Flash Lite support the Date class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
Date([yearOrTimevalue:Number], [month:Number], [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number])	Yes	Yes	Yes	Yes
getDate() : Number	Yes	Yes	Yes	Yes
getDay() : Number	Yes	Yes	Yes	Yes
getFullYear() : Number	Yes	Yes	Yes	Yes
getHours() : Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getLocaleLongDate() : String	No	Yes	Yes	Yes
getLocaleShortDate() : String	No	Yes	Yes	Yes
getLocaleTime() : String	No	Yes	Yes	Yes
getMilliseconds() : Number	Yes	Yes	Yes	Yes
getMinutes() : Number	Yes	Yes	Yes	Yes
getMonth() : Number	Yes	Yes	Yes	Yes
getSeconds() : Number	Yes	Yes	Yes	Yes
getTime() : Number	Yes	Yes	Yes	Yes
getTimezoneOffset() : Number	Yes	Yes	Yes	Yes
getUTCDate() : Number	Yes	Yes	Yes	Yes
getUTCDay() : Number	Yes	Yes	Yes	Yes
getUTCFullYear() : Number	Yes	Yes	Yes	Yes
getUTCHours() : Number	Yes	Yes	Yes	Yes
getUTCMilliseconds() : Number	Yes	Yes	Yes	Yes
getUTCMinutes() : Number	Yes	Yes	Yes	Yes
getUTCMonth() : Number	Yes	Yes	Yes	Yes
getUTCSeconds() : Number	Yes	Yes	Yes	Yes
getUTCYear() : Number	Yes	Yes	Yes	Yes
getYear() : Number	Yes	Yes	Yes	Yes
setDate(date:Number) : Number	Yes	Yes	Yes	Yes
setFullYear(year:Number, [month:Number], [date:Number]) : Number	Yes	Yes	Yes	Yes
setHours(hour:Number) : Number	Yes	Yes	Yes	Yes
setMilliseconds(millisecond:Number) : Number	Yes	Yes	Yes	Yes
setMinutes(minute:Number) : Number	Yes	Yes	Yes	Yes
setMonth(month:Number, [date:Number]) : Number	Yes	Yes	Yes	Yes
setSeconds(second:Number) : Number	Yes	Yes	Yes	Yes
setTime(millisecond:Number) : Number	Yes	Yes	Yes	Yes
setUTCDate(date:Number) : Number	Yes	Yes	Yes	Yes
setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
setUTCHours(hour:Number, [minute:Number], [second:Number], [millisecond:Number]) : Number	Yes	Yes	Yes	Yes
setUTCMilliseconds(millisecond:Number) : Number	Yes	Yes	Yes	Yes
setUTCMinutes(minute:Number, [second:Number], [millisecond:Number]) : Number	Yes	Yes	Yes	Yes
setUTCMonth(month:Number, [date:Number]) : Number	Yes	Yes	Yes	Yes
setUTCSeconds(second:Number, [millisecond:Number]) : Number	Yes	Yes	Yes	Yes
setYear(year:Number) : Number	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number]) : Number	Yes	Yes	Yes	Yes
valueOf() : Number	Yes	Yes	Yes	Yes

DisplacementMapFilter

The following table shows which versions of Flash Lite support the DisplacementMapFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alpha:Number	Yes	No	No	No
clone() : DisplacementMapFilter	Yes	No	No	No
color:Number	Yes	No	No	No
componentX:Number	Yes	No	No	No
componentY:Number	Yes	No	No	No
DisplacementMapFilter(mapBitmap:BitmapData, mapPoint:Point, componentX:Number, componentY:Number, scaleX:Number, scaleY:Number, [mode:String], [color:Number], [alpha:Number])	Yes	No	No	No
mapBitmap:BitmapData	Yes	No	No	No
mapPoint:Point	Yes	No	No	No
mode:String	Yes	No	No	No
scaleX:Number	Yes	No	No	No
scaleY:Number	Yes	No	No	No

DropShadowFilter

The following table shows which versions of Flash Lite support the DropShadowFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alpha:Number	Yes	No	No	No
angle:Number	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
clone() : DropShadowFilter	Yes	No	No	No
color:Number	Yes	No	No	No
distance:Number	Yes	No	No	No
DropShadowFilter([distance:Number], [angle:Number], [color:Number], [alpha:Number], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [inner:Boolean], [knockout:Boolean], [hideObject:Boolean])	Yes	No	No	No
hideObject:Boolean	Yes	No	No	No
inner:Boolean	Yes	No	No	No
knockout:Boolean	Yes	No	No	No
quality:Number	Yes	No	No	No
strength:Number	Yes	No	No	No

Error

The following table shows which versions of Flash Lite support the Error class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
Error([message:String])	Yes	Yes	Yes	Yes
message:String	Yes	Yes	Yes	Yes
name:String	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes

ExtendedKey

The following table shows which versions of Flash Lite support the ExtendedKey class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
SOFT1:String	No	Yes	Yes	Yes
SOFT10:String	No	Yes	Yes	Yes
SOFT11:String	No	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
SOFT12:String	No	Yes	Yes	Yes
SOFT2:String	No	Yes	Yes	Yes
SOFT3:String	No	Yes	Yes	Yes
SOFT4:String	No	Yes	Yes	Yes
SOFT5:String	No	Yes	Yes	Yes
SOFT6:String	No	Yes	Yes	Yes
SOFT7:String	No	Yes	Yes	Yes
SOFT8:String	No	Yes	Yes	Yes
SOFT9:String	No	Yes	Yes	Yes

ExternalInterface

The following table shows which versions of Flash Lite support the ExternalInterface class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addCallback(methodName:String, instance:Object, method:Function) : Boolean	Yes	No	Yes	Yes
available:Boolean [read-only]	Yes	No	Yes	Yes
call(methodName:String, [parameter1:Object]) : Object	Yes	No	Yes	Yes

FileReference

The following table shows which versions of Flash Lite support the FileReference class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
creationDate:Date [read-only]	Yes	No	No	No
creator:String [read-only]	Yes	No	No	No
modificationDate:Date [read-only]	Yes	No	No	No
name:String [read-only]	Yes	No	No	No
postData:String	Yes	No	No	No
size:Number [read-only]	Yes	No	No	No
type:String [read-only]	Yes	No	No	No

FileReferenceList

The following table shows which versions of Flash Lite support the FileReferenceList class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Void	Yes	No	No	No
browse([typelist:Array]) : Boolean	Yes	No	No	No
fileList:Array	Yes	No	No	No
FileReferenceList()	Yes	No	No	No
onCancel = function(fileRefList:FileReferenceList) {	Yes	No	No	No
onSelect = function(fileRefList:FileReferenceList) {	Yes	No	No	No
removeListener(listener:Object) : Boolean	Yes	No	No	No

Function

The following table shows which versions of Flash Lite support the Function class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
apply(thisObject:Object, [argArray:Array]) : Void	Yes	Yes	Yes	Yes
call(thisObject:Object, [parameter1:Object]) : Object	Yes	Deprecated	Deprecated	Deprecated

GlowFilter

The following table shows which versions of Flash Lite support the GlowFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alpha:Number	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
color:Number	Yes	No	No	No
inner:Boolean	Yes	No	No	No
knockout:Boolean	Yes	No	No	No
quality:Number	Yes	No	No	No
strength:Number	Yes	No	No	No

GradientBevelFilter

The following table shows which versions of Flash Lite support the GradientBevelFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alphas:Array	Yes	No	No	No
angle:Number	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
clone() : GradientBevelFilter	Yes	No	No	No
colors:Array	Yes	No	No	No
distance:Number	Yes	No	No	No
GradientBevelFilter([distance:Number], [angle:Number], [colors:Array], [alphas:Array], [ratios:Array], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])	Yes	No	No	No
knockout:Boolean	Yes	No	No	No
quality:Number	Yes	No	No	No
ratios:Array	Yes	No	No	No
strength:Number	Yes	No	No	No
type:String	Yes	No	No	No

GradientGlowFilter

The following table shows which versions of Flash Lite support the GradientFlowFilter class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
alphas:Array	Yes	No	No	No
angle:Number	Yes	No	No	No
blurX:Number	Yes	No	No	No
blurY:Number	Yes	No	No	No
colors:Array	Yes	No	No	No
distance:Number	Yes	No	No	No
GradientGlowFilter([distance:Number], [angle:Number], [colors:Array], [alphas:Array], [ratios:Array], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])	Yes	No	No	No
knockout:Boolean	Yes	No	No	No
quality:Number	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
ratios:Array	Yes	No	No	No
strength:Number	Yes	No	No	No
type:String	Yes	No	No	No

IME

The following table shows which versions of Flash Lite support the IME class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Void	Yes	No	No	No
ALPHANUMERIC_FULL:String	Yes	No	No	No
ALPHANUMERIC_HALF:String	Yes	No	No	No
CHINESE:String	Yes	No	No	No
doConversion() : Boolean	Yes	No	No	No
getConversionMode() : String	Yes	No	No	No
getEnabled() : Boolean	Yes	No	No	No
JAPANESE_HIRAGANA:String	Yes	No	No	No
JAPANESE_KATAKANA_FULL:String	Yes	No	No	No
JAPANESE_KATAKANA_HALF:String	Yes	No	No	No
KOREAN:String	Yes	No	No	No
onIMEComposition = function([readingString:String]) {}	Yes	No	No	No
removeListener(listener:Object) : Boolean	Yes	No	No	No
setCompositionString(composition:St ring) : Boolean	Yes	No	No	No
setConversionMode(mode:String) : Boolean	Yes	No	No	No
setEnabled(enabled:Boolean) : Boolean	Yes	No	No	No
UNKNOWN:String	Yes	No	No	No

Key

The following table shows which versions of Flash Lite support the Key class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_listeners:Array [read-only]	Yes	Yes	Yes	Yes
addListener(listener:Object) : Void	Yes	Yes	Yes	Yes
BACKSPACE:Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
CAPSLOCK:Number	Yes	Yes	Yes	Yes
CONTROL:Number	Yes	Yes	Yes	Yes
DELETEKEY:Number	Yes	Yes	Yes	Yes
DOWN:Number	Yes	Yes	Yes	Yes
END:Number	Yes	Yes	Yes	Yes
ENTER:Number	Yes	Yes	Yes	Yes
ESCAPE:Number	Yes	Yes	Yes	Yes
getAscii() : Number	Yes	Yes	Yes	Yes
getCode() : Number	Yes	Yes	Yes	Yes
HOME:Number	Yes	Yes	Yes	Yes
INSERT:Number	Yes	Yes	Yes	Yes
isAccessible() : Boolean	Yes	No	No	No
isDown(code:Number) : Boolean	Yes	Yes	Yes	Yes
isToggled(code:Number) : Boolean	Yes	No	No	No
LEFT:Number	Yes	Yes	Yes	Yes
onKeyDown = function() {}	Yes	Yes	Yes	Yes
onKeyUp = function() {}	Yes	Yes	Yes	Yes
PGDN:Number	Yes	Yes	Yes	Yes
PGUP:Number	Yes	Yes	Yes	Yes
removeListener(listener:Object) : Boolean	Yes	No	Yes	Yes
RIGHT:Number	Yes	Yes	Yes	Yes
SHIFT:Number	Yes	Yes	Yes	Yes
SPACE:Number	Yes	Yes	Yes	Yes
TAB:Number	Yes	Yes	Yes	Yes
UP:Number	Yes	Yes	Yes	Yes

LoadVars

The following table shows which versions of Flash Lite support the LoadVars class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addRequestHeader(header:Object, headerValue:String) : Void	Yes	Yes	Yes	Yes
contentType:String	Yes	Yes	Yes	Yes
decode(queryString:String) : Void	Yes	Yes	Yes	Yes
getBytesLoaded() : Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getBytesTotal() : Number	Yes	Yes	Yes	Yes
load(url:String) : Boolean	Yes	Yes	Yes	Yes
loaded:Boolean	Yes	Yes	Yes	Yes
LoadVars()	Yes	Yes	Yes	Yes
onData = function(src:String) {}	Yes	Yes	Yes	Yes
onHTTPStatus = function(httpStatus:Number) {}	Yes	No	No	Yes
onLoad = function(success:Boolean) {}	Yes	Yes	Yes	Yes
send(url:String, target:String, [method:String]) : Boolean	Yes	Yes	Yes	Yes
sendAndLoad(url:String, target:Object, [method:String]) : Boolean	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes

LocalConnection

The following table shows which versions of Flash Lite support the LocalConnection class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
allowDomain = function([sendingDomain:String]) {}	Yes	No	No	Yes
allowInsecureDomain = function([sendingDomain:String]) {}	Yes	No	No	Yes
close() : Void	Yes	No	No	Yes
connect(connectionName:String) : Boolean	Yes	No	No	Yes
domain() : String	Yes	No	No	Yes
LocalConnection()	Yes	No	No	Yes
onStatus = function(infoObject:Object) {}	Yes	No	No	Yes
send(connectionName:String, methodName:String, [args:Object]) : Boolean	Yes	No	No	Yes

Locale

The following table shows which versions of Flash Lite support the Locale class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addDelayedInstance(instance:Object, stringID:String) : Void	Yes	No	No	No
addXMLPath(langCode:String, path:String) : Void	Yes	No	No	No
autoReplace:Boolean	Yes	No	No	No
checkXMLStatus() : Boolean	Yes	No	No	No
getDefaultLang() : String	Yes	No	No	No
initialize() : Void	Yes	No	No	No
languageCodeArray:Array [read-only]	Yes	No	No	No
loadLanguageXML(xmlLanguageCode:String, customXmlCompleteCallback:Function) : Void	Yes	No	No	No
loadString(id:String) : String	Yes	No	No	No
loadStringEx(stringID:String, languageCode:String) : String	Yes	No	No	No
setDefaultLang(langCode:String) : Void	Yes	No	No	No
setLoadCallback(loadCallback:Function) : Void	Yes	No	No	No
setString(stringID:String, languageCode:String, stringValue:String) : Void	Yes	No	No	No
stringIDArray:Array [read-only]	Yes	No	No	No

Math

The following table shows which versions of Flash Lite support the Math class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
abs(x:Number) : Number	Yes	Yes	Yes	Yes
acos(x:Number) : Number	Yes	Yes	Yes	Yes
asin(x:Number) : Number	Yes	Yes	Yes	Yes
atan(tangent:Number) : Number	Yes	Yes	Yes	Yes
atan2(y:Number, x:Number) : Number	Yes	Yes	Yes	Yes
ceil(x:Number) : Number	Yes	Yes	Yes	Yes
cos(x:Number) : Number	Yes	Yes	Yes	Yes
E:Number	Yes	Yes	Yes	Yes
exp(x:Number) : Number	Yes	Yes	Yes	Yes
floor(x:Number) : Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
LN10:Number	Yes	Yes	Yes	Yes
LN2:Number	Yes	Yes	Yes	Yes
log(x:Number) : Number	Yes	Yes	Yes	Yes
LOG10E:Number	Yes	Yes	Yes	Yes
LOG2E:Number	Yes	Yes	Yes	Yes
max(x:Number, y:Number) : Number	Yes	Yes	Yes	Yes
min(x:Number, y:Number) : Number	Yes	Yes	Yes	Yes
PI:Number	Yes	Yes	Yes	Yes
pow(x:Number, y:Number) : Number	Yes	Yes	Yes	Yes
random() : Number	Yes	Yes	Yes	Yes
round(x:Number) : Number	Yes	Yes	Yes	Yes
sin(x:Number) : Number	Yes	Yes	Yes	Yes
sqrt(x:Number) : Number	Yes	Yes	Yes	Yes
SQRT1_2:Number	Yes	Yes	Yes	Yes
SQRT2:Number	Yes	Yes	Yes	Yes
tan(x:Number) : Number	Yes	Yes	Yes	Yes

Matrix

The following table shows which versions of Flash Lite support the Matrix class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
a:Number	Yes	No	No	Yes
b:Number	Yes	No	No	Yes
c:Number	Yes	No	No	Yes
clone() : Matrix	Yes	No	No	Yes
concat(m:Matrix) : Void	Yes	No	No	Yes
createBox(scaleX:Number, scaleY:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void	Yes	No	No	Yes
createGradientBox(width:Number, height:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void	Yes	No	No	No
d:Number	Yes	No	No	Yes
deltaTransformPoint(pt:Point) : Point	Yes	No	No	Yes
identity() : Void	Yes	No	No	Yes
invert() : Void	Yes	No	No	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
Matrix([a:Number], [b:Number], [c:Number], [d:Number], [tx:Number], [ty:Number])	Yes	No	No	Yes
rotate(angle:Number) : Void	Yes	No	No	Yes
scale(sx:Number, sy:Number) : Void	Yes	No	No	Yes
toString() : String	Yes	No	No	Yes
transformPoint(pt:Point) : Point	Yes	No	No	Yes
translate(tx:Number, ty:Number) : Void	Yes	No	No	Yes
tx:Number	Yes	No	No	Yes
ty:Number	Yes	No	No	Yes

Microphone

The following table shows which versions of Flash Lite support the Microphone class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
activityLevel:Number [read-only]	Yes	No	No	No
gain:Number [read-only]	Yes	No	No	No
get([index:Number]) : Microphone	Yes	No	No	No
index:Number [read-only]	Yes	No	No	No
muted:Boolean [read-only]	Yes	No	No	No
name:String [read-only]	Yes	No	No	No
names:Array [read-only]	Yes	No	No	No
onActivity = function(active:Boolean) {}	Yes	No	No	No
onStatus = function(infoObject:Object) {}	Yes	No	No	No
rate:Number [read-only]	Yes	No	No	No
setGain(gain:Number) : Void	Yes	No	No	No
setRate(rate:Number) : Void	Yes	No	No	No
setSilenceLevel(silenceLevel:Number, [timeOut:Number]) : Void	Yes	No	No	No
setUseEchoSuppression(useEchoSuppression:Boolean) : Void	Yes	No	No	No
silenceLevel:Number [read-only]	Yes	No	No	No
silenceTimeOut:Number [read-only]	Yes	No	No	No
useEchoSuppression:Boolean [read-only]	Yes	No	No	No

Mouse

The following table shows which versions of Flash Lite support the Mouse class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Void	Yes	Yes	Yes	Yes
hide() : Number	Yes	No	No	No
onMouseDown = function() {}	Yes	Yes	Yes	Yes
onMouseMove = function() {}	Yes	Yes	Yes	Yes
onMouseUp = function() {}	Yes	Yes	Yes	Yes
onMouseWheel = function([delta:Number], [scrollTarget:Object]) {}	Yes	No	No	No
removeListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
show() : Number	Yes	No	No	No

MovieClip

The following table shows which versions of Flash Lite support the MovieClip class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_alpha:Number	Yes	Yes	Yes	Yes
_currentframe:Number [read-only]	Yes	Yes	Yes	Yes
_droptarget:String [read-only]	Yes	Yes	Yes	Yes
_focusrect:Boolean	Yes	Yes	Yes	Yes
_framesloaded:Number [read-only]	Yes	Yes	Yes	Yes
_height:Number	Yes	Yes	Yes	Yes
_highquality:Number	Deprecated	Deprecated	Deprecated	Deprecated
_lockroot:Boolean	Yes	Yes	Yes	Yes
_name:String	Yes	Yes	Yes	Yes
_parent:MovieClip	Yes	Yes	Yes	Yes
_quality:String	Yes	Yes	Yes	Yes
_rotation:Number	Yes	Yes	Yes	Yes
_soundbuftime:Number	Yes	Yes	Yes	Yes
_target:String [read-only]	Yes	Yes	Yes	Yes
_totalframes:Number [read-only]	Yes	Yes	Yes	Yes
_url:String [read-only]	Yes	Yes	Yes	Yes
_visible:Boolean	Yes	Yes	Yes	Yes
_width:Number	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_x:Number	Yes	Yes	Yes	Yes
_xmouse:Number [read-only]	Yes	Yes	Yes	Yes
_xscale:Number	Yes	Yes	Yes	Yes
_y:Number	Yes	Yes	Yes	Yes
_ymouse:Number [read-only]	Yes	Yes	Yes	Yes
_yscale:Number	Yes	Yes	Yes	Yes
attachAudio(id:Object) : Void	Yes	No	No	No
attachBitmap(bmp:BitmapData, depth:Number, [pixelSnapping:String], [smoothing:Boolean]) : Void	Yes	No	No	Yes
attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip	Yes	Yes	Yes	Yes
beginBitmapFill(bmp:BitmapData, [matrix:Matrix], [repeat:Boolean], [smoothing:Boolean]) : Void	Yes	No	No	Yes
beginFill(rgb:Number, [alpha:Number]) : Void	Yes	Yes	Yes	Yes
beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object, [spreadMethod:String], [interpolationMethod:String], [focalPointRatio:Number]) : Void	Yes	No	No	No
blendMode:Object	Yes	No	No	No
cacheAsBitmap:Boolean	Yes	No	No	No
clear() : Void	Yes	Yes	Yes	Yes
createEmptyMovieClip(name:String, depth:Number) : MovieClip	Yes	Yes	Yes	Yes
createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number) : TextField	Yes	Yes	Yes	Yes
curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number) : Void	Yes	Yes	Yes	Yes
duplicateMovieClip(name:String, depth:Number, [initObject:Object]) : MovieClip	Yes	Yes	Yes	Yes
enabled:Boolean	Yes	Yes	Yes	Yes
endFill() : Void	Yes	Yes	Yes	Yes
filters:Array	Yes	No	No	No
focusEnabled:Boolean	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
forceSmoothing:Boolean	Yes	No	No	No
getBounds(bounds:Object) : Object	Yes	Yes	Yes	Yes
getBytesLoaded() : Number	Yes	Yes	Yes	Yes
getBytesTotal() : Number	Yes	Yes	Yes	Yes
getDepth() : Number	Yes	Yes	Yes	Yes
getInstanceAtDepth(depth:Number) : MovieClip	Yes	Yes	Yes	Yes
getNextHighestDepth() : Number	Yes	Yes	Yes	Yes
getRect(bounds:Object) : Object	Yes	No	No	No
getSWFVersion() : Number	Yes	Yes	Yes	Yes
getTextSnapshot() : TextSnapshot	Yes	No	No	No
getURL(url:String, [window:String], [method:String]) : Void	Yes	Yes	Yes	Yes
globalToLocal(pt:Object) : Void	Yes	Yes	Yes	Yes
gotoAndPlay(frame:Object) : Void	Yes	Yes	Yes	Yes
gotoAndStop(frame:Object) : Void	Yes	Yes	Yes	Yes
hitArea:Object	Yes	Yes	Yes	Yes
hitTest() : Boolean	Yes	Yes	Yes	Yes
lineGradientStyle(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object, [spreadMethod:String], [interpolationMethod:String], [focalPointRatio:Number]) : Void	Yes	No	No	No
lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number) : Void	Yes	Yes	Yes	Yes
lineTo(x:Number, y:Number) : Void	Yes	Yes	Yes	Yes
loadMovie(url:String, [method:String]) : Void	Yes	Yes	Yes	Yes
loadVariables(url:String, [method:String]) : Void	Yes	Yes	Yes	Yes
localToGlobal(pt:Object) : Void	Yes	Yes	Yes	Yes
menu:ContextMenu	Yes	No	No	No
moveTo(x:Number, y:Number) : Void	Yes	Yes	Yes	Yes
nextFrame() : Void	Yes	Yes	Yes	Yes
onData = function() {}	Yes	Yes	Yes	Yes
onDragOut = function() {}	Yes	Yes	Yes	Yes
onDragOver = function() {}	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
onEnterFrame = function() {}	Yes	Yes	Yes	Yes
onKeyDown = function() {}	Yes	Yes	Yes	Yes
onKeyUp = function() {}	Yes	Yes	Yes	Yes
onKillFocus = function(newFocus:Object) {}	Yes	Yes	Yes	Yes
onLoad = function() {}	Yes	Yes	Yes	Yes
onMouseDown = function() {}	Yes	Yes	Yes	Yes
onMouseMove = function() {}	Yes	Yes	Yes	Yes
onMouseUp = function() {}	Yes	Yes	Yes	Yes
onPress = function() {}	Yes	Yes	Yes	Yes
onRelease = function() {}	Yes	Yes	Yes	Yes
onReleaseOutside = function() {}	Yes	Yes	Yes	Yes
onRollOut = function() {}	Yes	Yes	Yes	Yes
onRollOver = function() {}	Yes	Yes	Yes	Yes
onSetFocus = function(oldFocus:Object) {}	Yes	Yes	Yes	Yes
onUnload = function() {}	Yes	Yes	Yes	Yes
opaqueBackground:Number	Yes	No	No	No
play() : Void	Yes	Yes	Yes	Yes
prevFrame() : Void	Yes	Yes	Yes	Yes
removeMovieClip() : Void	Yes	Yes	Yes	Yes
scale9Grid:Rectangle	Yes	No	No	No
scrollRect:Object	Yes	No	No	No
setMask(mc:Object) : Void	Yes	Yes	Yes	Yes
startDrag([lockCenter:Boolean], [left:Number], [top:Number], [right:Number], [bottom:Number]) : Void	Yes	Yes	Yes	Yes
stop() : Void	Yes	Yes	Yes	Yes
stopDrag() : Void	Yes	Yes	Yes	Yes
swapDepths(target:Object) : Void	Yes	Yes	Yes	Yes
tabChildren:Boolean	Yes	Yes	Yes	Yes
tabEnabled:Boolean	Yes	Yes	Yes	Yes
tabIndex:Number	Yes	Yes	Yes	Yes
trackAsMenu:Boolean	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
transform:Transform	Yes	No	No	No
unloadMovie() : Void	Yes	Yes	Yes	Yes
useHandCursor:Boolean	Yes	No	No	No

MovieClipLoader

The following table shows which versions of Flash Lite support the MovieClipLoader class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
checkPolicyFile:Boolean	Yes	No	No	No
getProgress(target:Object) : Object	Yes	Yes	Yes	Yes
loadClip(url:String, target:Object) : Boolean	Yes	Yes	Yes	Yes
MovieClipLoader()	Yes	Yes	Yes	Yes
onLoadComplete = function([target_mc:MovieClip], [httpStatus:Number]) {}	Yes	Yes	Yes	Yes
onLoadError = function(target_mc:MovieClip, errorCode:String, [httpStatus:Number]) {}	Yes	Yes	Yes	Yes
onLoadInit = function([target_mc:MovieClip]) {}	Yes	Yes	Yes	Yes
onLoadProgress = function([target_mc:MovieClip], loadedBytes:Number, totalBytes:Number) {}	Yes	Yes	Yes	Yes
onLoadStart = function([target_mc:MovieClip]) {}	Yes	Yes	Yes	Yes
removeListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
unloadClip(target:Object) : Boolean	Yes	Yes	Yes	Yes

NetConnection

The following table shows which versions of Flash Lite support the NetConnection class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
connect(targetURI:String) : Boolean	Yes	No	Yes	Yes
NetConnection()	Yes	No	Yes	Yes

NetStream

The following table shows which versions of Flash Lite support the NetStream class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
bufferLength:Number [read-only]	Yes	No	Yes	Yes
bufferTime:Number [read-only]	Yes	No	Yes	Yes
bytesLoaded:Number [read-only]	Yes	No	Yes	Yes
bytesTotal:Number [read-only]	Yes	No	Yes	Yes
checkPolicyFile:Boolean	Yes	No	No	No
close() : Void	Yes	No	Yes	Yes
currentFps:Number [read-only]	Yes	No	Yes	Yes
NetStream(connection:NetConnection)	Yes	No	Yes	Yes
onCuePoint = function(infoObject:Object) {}	Yes	No	Yes	Yes
onMetaData = function(infoObject:Object) {}	Yes	No	Yes	Yes
onStatus = function(infoObject:Object) {}	Yes	No	Yes	Yes
pause([flag:Boolean]) : Void	Yes	No	Yes	Yes
play(name:Object, start:Number, len:Number, reset:Object) : Void	Yes	No	Yes	Yes
seek(offset:Number) : Void	Yes	No	Yes	Yes
setBufferTime(bufferTime:Number) : Void	Yes	No	Yes	Yes
time:Number [read-only]	Yes	No	Yes	Yes

Number

The following table shows which versions of Flash Lite support the Number class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
MAX_VALUE:Number	Yes	Yes	Yes	Yes
MIN_VALUE:Number	Yes	Yes	Yes	Yes
NaN:Number	Yes	Yes	Yes	Yes
NEGATIVE_INFINITY:Number	Yes	Yes	Yes	Yes
Number(num:Object)	Yes	Yes	Yes	Yes
POSITIVE_INFINITY:Number	Yes	Yes	Yes	Yes
toString(radix:Number) : String	Yes	Yes	Yes	Yes
valueOf() : Number	Yes	Yes	Yes	Yes

Object

The following table shows which versions of Flash Lite support the Object class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
__proto__:Object	Yes	Yes	Yes	Yes
__resolve:Object	Yes	Yes	Yes	Yes
addProperty(name:String, getter:Function, setter:Function) : Boolean	Yes	Yes	Yes	Yes
constructor:Object	Yes	Yes	Yes	Yes
hasOwnProperty(name:String) : Boolean	Yes	Yes	Yes	Yes
isPropertyEnumerable(name:String) : Boolean	Yes	Yes	Yes	Yes
isPrototypeOf(theClass:Object) : Boolean	Yes	Yes	Yes	Yes
Object()	Yes	Yes	Yes	Yes
prototype:Object	Yes	Yes	Yes	Yes
registerClass(name:String, theClass:Function) : Boolean	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
unwatch(name:String) : Boolean	Yes	Yes	Yes	Yes
valueOf() : Object	Yes	Yes	Yes	Yes
watch(name:String, callback:Function, [userData:Object]) : Boolean	Yes	Yes	Yes	Yes

Point

The following table shows which versions of Flash Lite support the Point class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
add(v:Point) : Point	Yes	No	No	Yes
clone() : Point	Yes	No	No	Yes
distance(pt1:Point, pt2:Point) : Number	Yes	No	No	Yes
equals(toCompare:Object) : Boolean	Yes	No	No	Yes
interpolate(pt1:Point, pt2:Point, f:Number) : Point	Yes	No	No	Yes
length:Number	Yes	No	No	Yes
normalize(length:Number) : Void	Yes	No	No	Yes
offset(dx:Number, dy:Number) : Void	Yes	No	No	Yes
Point(x:Number, y:Number)	Yes	No	No	Yes
polar(len:Number, angle:Number) : Point	Yes	No	No	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
subtract(v:Point) : Point	Yes	No	No	Yes
toString() : String	Yes	No	No	Yes
x:Number	Yes	No	No	Yes
y:Number	Yes	No	No	Yes

PrintJob

The following table shows which versions of Flash Lite support the PrintJob class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addPage(target:Object, [printArea:Object], [options:Object], [frameNum:Number]) : Boolean	Yes	No	No	No
orientation:String [read-only]	Yes	No	No	No
pageHeight:Number [read-only]	Yes	No	No	No
pageWidth:Number [read-only]	Yes	No	No	No
paperHeight:Number [read-only]	Yes	No	No	No
paperWidth:Number [read-only]	Yes	No	No	No
PrintJob()	Yes	No	No	No
send() : Void	Yes	No	No	No
start() : Boolean	Yes	No	No	No

Rectangle

The following table shows which versions of Flash Lite support the Rectangle class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
bottom:Number	Yes	No	No	Yes
bottomRight:Point	Yes	No	No	Yes
clone() : Rectangle	Yes	No	No	Yes
contains(x:Number, y:Number) : Boolean	Yes	No	No	Yes
containsPoint(pt:Point) : Boolean	Yes	No	No	Yes
containsRectangle(rect:Rectangle) : Boolean	Yes	No	No	Yes
equals(toCompare:Object) : Boolean	Yes	No	No	Yes
height:Number	Yes	No	No	Yes
inflate(dx:Number, dy:Number) : Void	Yes	No	No	Yes
inflatePoint(pt:Point) : Void	Yes	No	No	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
intersection(toIntersect:Rectangle) : Rectangle	Yes	No	No	Yes
intersects(toIntersect:Rectangle) : Boolean	Yes	No	No	Yes
isEmpty() : Boolean	Yes	No	No	Yes
left:Number	Yes	No	No	Yes
offset(dx:Number, dy:Number) : Void	Yes	No	No	Yes
offsetPoint(pt:Point) : Void	Yes	No	No	Yes
Rectangle(x:Number, y:Number, width:Number, height:Number)	Yes	No	No	Yes
right:Number	Yes	No	No	Yes
setEmpty() : Void	Yes	No	No	Yes
size:Point	Yes	No	No	Yes
top:Number	Yes	No	No	Yes
topLeft:Point	Yes	No	No	Yes
toString() : String	Yes	No	No	Yes
union(toUnion:Rectangle) : Rectangle	Yes	No	No	Yes
width:Number	Yes	No	No	Yes
x:Number	Yes	No	No	Yes
y:Number	Yes	No	No	Yes

security

The following table shows which versions of Flash Lite support the security class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
allowDomain(domain1:String) : Void	Yes	Yes	Yes	Yes
allowInsecureDomain(domain:String) : Void	Yes	Yes	Yes	Yes
loadPolicyFile(url:String) : Void	Yes	Yes	Yes	Yes
sandboxType:String [read-only]	Yes	No	No	Yes

Selection

The following table shows which versions of Flash Lite support the Selection class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Void	Yes	Yes	Yes	Yes
getBeginIndex() : Number	Yes	No	No	No
getCaretIndex() : Number	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getEndIndex() : Number	Yes	No	No	No
getFocus() : String	Yes	Yes	Yes	Yes
onSetFocus = function([oldfocus:Object], [newfocus:Object]) {}	Yes	Yes	Yes	Yes
removeListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
setFocus(newFocus:Object) : Boolean	Yes	Yes	Yes	Yes
setSelection(beginIndex:Number, endIndex:Number) : Void	Yes	No	No	No

SharedObject

The following table shows which versions of Flash Lite support the SharedObject class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(objectName:String, notifyFunction:Function) : Void	No	Yes	Yes	Yes
clear() : Void	Yes	Yes	Yes	Yes
data:Object	Yes	Yes	Yes	Yes
flush([minDiskSpace:Number]) : Object	Yes	Partial	Partial	Partial
getLocal(name:String, [localPath:String], [secure:Boolean]) : SharedObject	Yes	Partial	Partial	Partial
getMaxSize() : Number	No	Yes	Yes	Yes
getSize() : Number	Yes	Yes	Yes	Yes
onStatus = function(infoObject:Object) {}	Yes	Yes	Yes	Yes
removeListener(objectName:String)	No	Yes	Yes	Yes

Sound

The following table shows which versions of Flash Lite support the Sound class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
attachSound(id:String) : Void	Yes	Yes	Yes	Yes
checkPolicyFile:Boolean	Yes	No	No	No
duration:Number [read-only]	Yes	Partial	Partial	Partial
getBytesLoaded() : Number	Yes	Yes	Yes	Yes
getBytesTotal() : Number	Yes	Yes	Yes	Yes
getPan() : Number	Yes	Partial	Partial	Partial

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getTransform() : Object	Yes	Partial	Partial	Partial
getVolume() : Number	Yes	Yes	Yes	Yes
id3:Object [read-only]	Yes	Yes	Yes	Yes
loadSound(url:String, isStreaming:Boolean) : Void	Yes	Partial	Partial	Partial
onID3 = function() {}	Yes	Yes	Yes	Yes
onLoad = function(success:Boolean) {}	Yes	Yes	Yes	Yes
onSoundComplete = function() {}	Yes	Yes	Yes	Yes
position:Number [read-only]	Yes	Partial	Partial	Partial
setPan(value:Number) : Void	Yes	Partial	Partial	Partial
setTransform(transformObject:Object) : Void	Yes	Partial	Partial	Partial
setVolume(value:Number) : Void	Yes	Partial	Partial	Partial
Sound([target:Object])	Yes	Yes	Yes	Yes
start([secondOffset:Number], [loops:Number]) : Void	Yes	Yes	Yes	Yes
stop([linkageID:String]) : Void	Yes	Yes	Yes	Yes

Stage

The following table shows which versions of Flash Lite support the Stage class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addListener(listener:Object) : Void	Yes	Yes	Yes	Yes
align:String	Yes	Yes	Yes	Yes
displayState:String	Yes	No	No	No
height:Number	Yes	Yes	Yes	Yes
onFullScreen = function(bFull:Boolean) {}	Yes	No	No	No
onResize = function() {}	Yes	Yes	Yes	Yes
removeListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
scaleMode:String	Yes	Yes	Yes	Yes
showMenu:Boolean	Yes	No	No	No
width:Number	Yes	Yes	Yes	Yes

String

The following table shows which versions of Flash Lite support the String class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
charAt(index:Number) : String	Yes	Yes	Yes	Yes
charCodeAt(index:Number) : Number	Yes	Yes	Yes	Yes
concat(value:Object) : String	Yes	Yes	Yes	Yes
fromCharCode() : String	Yes	Yes	Yes	Yes
indexOf(value:String, [startIndex:Number]) : Number	Yes	Yes	Yes	Yes
lastIndexOf(value:String, [startIndex:Number]) : Number	Yes	Yes	Yes	Yes
length:Number	Yes	Yes	Yes	Yes
slice(start:Number, end:Number) : String	Yes	Yes	Yes	Yes
split(delimiter:String, [limit:Number]) : Array	Yes	Yes	Yes	Yes
String(value:String)	Yes	Yes	Yes	Yes
substr(start:Number, length:Number) : String	Yes	Yes	Yes	Yes
substring(start:Number, end:Number) : String	Yes	Yes	Yes	Yes
toLowerCase() : String	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
toUpperCase() : String	Yes	Yes	Yes	Yes
valueOf() : String	Yes	Yes	Yes	Yes

StyleSheet

The following table shows which versions of Flash Lite support the StyleSheet class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
clear() : Void	Yes	No	No	Yes
getStyle(name:String) : Object	Yes	No	No	Yes
getStyleNames() : Array	Yes	No	No	Yes
load(url:String) : Boolean	Yes	No	No	Yes
onLoad = function(success:Boolean) {}	Yes	No	No	Yes
parseCSS(cssText:String) : Boolean	Yes	No	No	Yes
setStyle(name:String, style:Object) : Void	Yes	No	No	Yes
StyleSheet()	Yes	No	No	Yes
transform(style:Object) : TextFormat	Yes	No	No	Yes

System

The following table shows which versions of Flash Lite support the System class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
exactSettings:Boolean	Yes	No	No	No
onStatus = function(infoObject:Object) {}	Yes	Yes	Yes	Yes
setClipboard(text:String) : Void	Yes	No	No	No
showSettings([tabID:Number]) : Void	Yes	No	No	No
useCodepage:Boolean	Yes	Yes	Yes	Yes

TextField

The following table shows which versions of Flash Lite support the TextField class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_alpha:Number	Yes	Yes	Yes	Yes
_height:Number	Yes	Yes	Yes	Yes
_highquality:Number	Deprecated	Deprecated	Deprecated	Deprecated
_name:String	Yes	Yes	Yes	Yes
_parent:MovieClip	Yes	Yes	Yes	Yes
_quality:String	Yes	Yes	Yes	Yes
_rotation:Number	Yes	Yes	Yes	Yes
_soundbuftime:Number	Yes	Yes	Yes	Yes
_target:String [read-only]	Yes	Yes	Yes	Yes
_url:String [read-only]	Yes	Yes	Yes	Yes
_visible:Boolean	Yes	Yes	Yes	Yes
_width:Number	Yes	Yes	Yes	Yes
_x:Number	Yes	Yes	Yes	Yes
_xmouse:Number [read-only]	Yes	Yes	Yes	Yes
_xscale:Number	Yes	Yes	Yes	Yes
_y:Number	Yes	Yes	Yes	Yes
_ymouse:Number [read-only]	Yes	Yes	Yes	Yes
_yscale:Number	Yes	Yes	Yes	Yes
addListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
antiAliasType:String	Yes	No	No	No
autoSize:Object	Yes	Yes	Yes	Yes
background:Boolean	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
backgroundColor:Number	Yes	Yes	Yes	Yes
border:Boolean	Yes	Yes	Yes	Yes
borderColor:Number	Yes	Yes	Yes	Yes
bottomScroll:Number [read-only]	Yes	Yes	Yes	Yes
condenseWhite:Boolean	Yes	Yes	Yes	Yes
embedFonts:Boolean	Yes	Yes	Yes	Yes
filters:Array	Yes	No	No	No
getDepth() : Number	Yes	Yes	Yes	Yes
getFontList() : Array	Yes	No	No	No
getNewTextFormat() : TextFormat	Yes	Yes	Yes	Yes
getTextFormat([beginIndex:Number, endIndex:Number]) : TextFormat	Yes	Yes	Yes	Yes
gridFitType:String	Yes	No	No	No
hscroll:Number	Yes	Yes	Yes	Yes
html:Boolean	Yes	Yes	Yes	Yes
htmlText:String	Yes	Yes	Yes	Yes
length:Number [read-only]	Yes	Yes	Yes	Yes
maxChars:Number	Yes	Yes	Yes	Yes
maxhscroll:Number [read-only]	Yes	Yes	Yes	Yes
maxscroll:Number [read-only]	Yes	Yes	Yes	Yes
menu:ContextMenu	Yes	No	No	No
mouseWheelEnabled:Boolean	Yes	No	No	No
multiline:Boolean	Yes	Yes	Yes	Yes
onChanged = function(changedField:TextField) {}	Yes	Yes	Yes	Yes
onKillFocus = function(newFocus:Object) {}	Yes	Yes	Yes	Yes
onScroller = function(scrolledField:TextField) {}	Yes	Yes	Yes	Yes
onSetFocus = function(oldFocus:Object) {}	Yes	Yes	Yes	Yes
password:Boolean	Yes	Yes	Yes	Yes
removeListener(listener:Object) : Boolean	Yes	Yes	Yes	Yes
removeTextField() : Void	Yes	Yes	Yes	Yes
replaceSet(newText:String) : Void	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
replaceText(beginIndex:Number, endIndex:Number, newText:String) : Void	Yes	Yes	Yes	Yes
restrict:String	Yes	No	No	No
scroll:Number	Yes	Yes	Yes	Yes
selectable:Boolean	Yes	Yes	Yes	Yes
setNewTextFormat(tf:TextFormat) : Void	Yes	Yes	Yes	Yes
setTextFormat([beginIndex:Number], [endIndex:Number], textFormat:TextFormat) : Void	Yes	Yes	Yes	Yes
sharpness:Number	Yes	No	No	No
styleSheet:StyleSheet	Yes	No	No	No
tabEnabled:Boolean	Yes	Yes	Yes	Yes
tabIndex:Number	Yes	Yes	Yes	Yes
text:String	Yes	Yes	Yes	Yes
textColor:Number	Yes	Yes	Yes	Yes
textHeight:Number	Yes	Yes	Yes	Yes
textWidth:Number	Yes	Yes	Yes	Yes
thickness:Number	Yes	No	No	No
type:String	Yes	Yes	Yes	Yes
variable:String	Yes	Yes	Yes	Yes
wordWrap:Boolean	Yes	Yes	Yes	Yes

TextFormat

The following table shows which versions of Flash Lite support the TextFormat class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
align:String	Yes	Yes	Yes	Yes
blockIndent:Number	Yes	Yes	Yes	Yes
bold:Boolean	Yes	Yes	Yes	Yes
bullet:Boolean	Yes	Partial	Partial	Partial
color:Number	Yes	Yes	Yes	Yes
font:String	Yes	Partial	Partial	Partial
getTextExtent(text:String, [width:Number]) : Object	Yes	Yes	Yes	Yes
indent:Number	Yes	Yes	Yes	Yes
italic:Boolean	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
kerning:Boolean	Yes	No	No	No
leading:Number	Yes	Yes	Yes	Yes
leftMargin:Number	Yes	Yes	Yes	Yes
letterSpacing:Number	Yes	No	No	No
rightMargin:Number	Yes	Yes	Yes	Yes
size:Number	Yes	Yes	Yes	Yes
tabStops:Array	Yes	Partial	Partial	Partial
target:String	Yes	No	No	No
TextFormat([font:String], [size:Number], [color:Number], [bold:Boolean], [italic:Boolean], [underline:Boolean], [url:String], [target:String], [align:String], [leftMargin:Number], [rightMargin:Number], [indent:Number], [leading:Number])	Yes	Yes	Yes	Yes
underline:Boolean	Yes	Yes	Yes	Yes
url:String	Yes	No	No	No

TextRenderer

The following table shows which versions of Flash Lite support the TextRenderer class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
displayMode:String	Yes	No	No	No
maxLevel:Number	Yes	No	No	No
setAdvancedAntialiasingTable(fontName:String, fontStyle:String, colorType:String, advancedAntialiasingTable:Array) : Void	Yes	No	No	No

TextSnapshot

The following table shows which versions of Flash Lite support the TextSnapshot class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
findText(startIndex:Number, textToFind:String, caseSensitive:Boolean) : Number	Yes	No	No	No
getCount() : Number	Yes	No	No	No
getSelected(start:Number, [end:Number]) : Boolean	Yes	No	No	No
getSelectedText([includeLineEndings: Boolean]) : String	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
getText(start:Number, end:Number, [includeLineEndings:Boolean]) : String	Yes	No	No	No
getTextRunInfo(beginIndex:Number, endIndex:Number) : Array	Yes	No	No	No
hitTestTextNearPos(x:Number, y:Number, [closeDist:Number]) : Number	Yes	No	No	No
setSelectColor(color:Number) : Void	Yes	No	No	No
setSelected(start:Number, end:Number, select:Boolean) : Void	Yes	No	No	No

Transform

The following table shows which versions of Flash Lite support the Transform class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
colorTransform:ColorTransform	Yes	No	No	Yes
concatenatedColorTransform:ColorTransform [read-only]	Yes	No	No	Yes
concatenatedMatrix:Matrix [read-only]	Yes	No	No	Yes
matrix:Matrix	Yes	No	No	Yes
pixelBounds:Rectangle	Yes	No	No	Yes
Transform(mc:MovieClip)	Yes	No	No	Yes

Video

The following table shows which versions of Flash Lite support the Video class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_alpha:Number	Yes	No	No	No
_height:Number	Yes	No	No	No
_name:String	Yes	No	No	No
_parent:MovieClip	Yes	No	No	No
_rotation:Number	Yes	No	No	No
_visible:Boolean	Yes	No	No	No
_width:Number	Yes	No	No	No
_x:Number	Yes	No	No	No
_xmouse:Number [read-only]	Yes	No	No	No
_xscale:Number	Yes	No	No	No
_y:Number	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
_mouse:Number [read-only]	Yes	No	No	No
_yscale:Number	Yes	No	No	No
attachVideo(source:Object) : Void	Yes	No	Yes	Yes
clear() : Void	Yes	No	Yes	Yes
close() : Void	No	Yes	Yes	Yes
deblocking:Number	Yes	No	No	Yes
height:Number [read-only]	Yes	No	Yes	Yes
onStatus = function(info:Object) {}	No	Yes	Yes	Yes
pause() : Void	No	Yes	Yes	Yes
play() : Boolean	No	Yes	Yes	Yes
resume() : Void	No	Yes	Yes	Yes
smoothing:Boolean	Yes	No	No	Yes
stop() : Void	No	Yes	Yes	Yes
width:Number [read-only]	Yes	No	Yes	Yes

XML

The following table shows which versions of Flash Lite support the XML class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
addRequestHeader(header:Object, headerValue:String) : Void	Yes	Yes	Yes	Yes
contentType:String	Yes	Yes	Yes	Yes
createElement(name:String) : XMLNode	Yes	Yes	Yes	Yes
createTextNode(value:String) : XMLNode	Yes	Yes	Yes	Yes
docTypeDecl:String	Yes	Yes	Yes	Yes
getBytesLoaded() : Number	Yes	Yes	Yes	Yes
getBytesTotal() : Number	Yes	Yes	Yes	Yes
idMap:Object	Yes	No	No	No
ignoreWhite:Boolean	Yes	Yes	Yes	Yes
load(url:String) : Boolean	Yes	Yes	Yes	Yes
loaded:Boolean	Yes	Yes	Yes	Yes
onData = function(src:String) {}	Yes	Yes	Yes	Yes
onHTTPStatus = function(httpStatus:Number) {}	Yes	No	No	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
onLoad = function(success:Boolean) {}	Yes	No	No	Yes
parseXML(value:String) : Void	Yes	Yes	Yes	Yes
send(url:String, [target:String], [method:String]) : Boolean	Yes	Yes	Yes	Yes
sendAndLoad(url:String, resultXML:XML) : Void	Yes	Yes	Yes	Yes
status:Number	Yes	Yes	Yes	Yes
XML(text:String)	Yes	Yes	Yes	Yes
xmlDecl:String	Yes	Yes	Yes	Yes

XMLNode

The following table shows which versions of Flash Lite support the XMLNode class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
appendChild(newChild:XMLNode) : Void	Yes	Yes	Yes	Yes
attributes:Object	Yes	Yes	Yes	Yes
childNodes:Array [read-only]	Yes	Yes	Yes	Yes
cloneNode(deep:Boolean) : XMLNode	Yes	Yes	Yes	Yes
firstChild:XMLNode [read-only]	Yes	Yes	Yes	Yes
getNamespaceForPrefix(prefix:String) : String	Yes	No	No	No
getPrefixForNamespace(nsURI:String) : String	Yes	No	No	No
hasChildNodes() : Boolean	Yes	Yes	Yes	Yes
insertBefore(newChild:XMLNode, insertPoint:XMLNode) : Void	Yes	Yes	Yes	Yes
lastChild:XMLNode [read-only]	Yes	Yes	Yes	Yes
localName:String [read-only]	Yes	No	No	No
namespaceURI:String [read-only]	Yes	No	No	No
nextSibling:XMLNode [read-only]	Yes	Yes	Yes	Yes
nodeName:String	Yes	Yes	Yes	Yes
nodeType:Number [read-only]	Yes	Yes	Yes	Yes
nodeValue:String	Yes	Yes	Yes	Yes
parentNode:XMLNode [read-only]	Yes	Yes	Yes	Yes
prefix:String [read-only]	Yes	No	No	No
previousSibling:XMLNode [read-only]	Yes	Yes	Yes	Yes

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
removeNode() : Void	Yes	Yes	Yes	Yes
toString() : String	Yes	Yes	Yes	Yes
XMLNode(type:Number, value:String)	Yes	Yes	Yes	Yes

XMLSocket

The following table shows which versions of Flash Lite support the XMLSocket class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
close() : Void	Yes	Yes	Yes	Yes
connect(url:String, port:Number) : Boolean	Yes	Yes	Yes	Yes
onClose = function() {}	Yes	Yes	Yes	Yes
onConnect = function(success:Boolean) {}	Yes	Yes	Yes	Yes
onData = function(src:String) {}	Yes	Yes	Yes	Yes
onXML = function(src:XML) {}	Yes	Yes	Yes	Yes
send(data:Object) : Void	Yes	Yes	Yes	Yes
XMLSocket()	Yes	Yes	Yes	Yes

XMLUI

The following table shows which versions of Flash Lite support the XMLUI class and its members.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
accept() : Void	Yes	No	No	No
cancel() : Void	Yes	No	No	No
get(name:String) : String	Yes	No	No	No
set(name:String, value:String) : Void	Yes	No	No	No

Global functions and properties

The following table shows which versions of Flash Lite support the various global functions and properties.

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
add	No	Yes	Yes	Yes
and	No	Yes	Yes	Yes
Array([numElements:Number], [elementN:Object])	Yes	Yes	Yes	Yes
asfunction(function:String, parameter:String)	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
Boolean(expression:Object)	Yes	Yes	Yes	Yes
break	No	Yes	Yes	Yes
call(frame:Object)	Deprecated	Deprecated	Deprecated	Deprecated
case	No	Yes	Yes	Yes
chr(number:Number)	Deprecated	Deprecated	Deprecated	Deprecated
clearInterval(intervalID:Number)	Yes	Yes	Yes	Yes
continue	No	Yes	Yes	Yes
default	No	Yes	Yes	Yes
do while	No	Yes	Yes	Yes
duplicateMovieClip(target:Object, newname:String, depth:Number)	Yes	Yes	Yes	Yes
else	No	Yes	Yes	Yes
else if	No	Yes	Yes	Yes
eq	No	Yes	Yes	Yes
escape(expression:String)	Yes	Yes	Yes	Yes
eval(expression:Object)	Yes	Yes	Yes	Yes
0	No	Yes	Yes	Yes
for	No	Yes	Yes	Yes
for ... in	No	Yes	Yes	Yes
fscommand(command:String, parameters:String)	Yes	Yes	Yes	Yes
fscommand2(command:String, parameters:String)	No	Yes	Yes	Yes
function	No	Yes	Yes	Yes
getProperty(my_mc:String, property)	Deprecated	Yes	Yes	Yes
getTimer()	Yes	Yes	Yes	Yes
getURL(url:String, [window:String], [method:String])	Yes	Yes	Yes	Yes
getVersion()	Yes	Yes	Yes	Yes
gotoAndPlay([scene:String], frame:Object)	Yes	Yes	Yes	Yes
gotoAndStop([scene:String], frame:Object)	Yes	Yes	Yes	Yes
if	No	Yes	Yes	Yes
ifFrameLoaded([scene:String], frame:Object)	Deprecated	Deprecated	Deprecated	Deprecated
int(value:Number)	Deprecated	Deprecated	Deprecated	Deprecated

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
isFinite(expression:Object)	Yes	Yes	Yes	Yes
isNaN(expression:Object)	Yes	Yes	Yes	Yes
length(expression:String, variable:Object)	Deprecated	Deprecated	Deprecated	Deprecated
loadMovie(url:String, target:Object, [method:String])	Yes	Yes	Yes	Yes
loadMovieNum(url:String, level:Number, [method:String])	Yes	Yes	Yes	Yes
loadVariables(url:String, target:Object, [method:String])	Yes	Yes	Yes	Yes
loadVariablesNum(url:String, level:Number, [method:String])	Yes	Yes	Yes	Yes
mbchr(number:Number)	Deprecated	Deprecated	Deprecated	Deprecated
mblength(string:String)	Deprecated	Deprecated	Deprecated	Deprecated
mbord(character:String)	Deprecated	Deprecated	Deprecated	Deprecated
mbsubstring(value:String, index:Number, count:Number)	Deprecated	Deprecated	Deprecated	Deprecated
MMExecute(command:String)	Yes	No	No	No
ne	No	Yes	Yes	Yes
new	No	Yes	Yes	Yes
newline	No	Yes	Yes	Yes
nextFrame()	Yes	Yes	Yes	Yes
nextScene()	Yes	Yes	Yes	Yes
not	No	Yes	Yes	Yes
Number(expression:Object)	Yes	Yes	Yes	Yes
Object([value:Object])	Yes	Yes	Yes	Yes
on(mouseEvent:Object)	Yes	Yes	Yes	Yes
onClipEvent(movieEvent:Object)	Yes	Yes	Yes	Yes
ord(character:String)	Deprecated	Deprecated	Deprecated	Deprecated
parseFloat(string:String)	Yes	Yes	Yes	Yes
parseInt(expression:String, [radix:Number])	Yes	Yes	Yes	Yes
play()	Yes	Yes	Yes	Yes
prevFrame()	Yes	Yes	Yes	Yes
prevScene()	Yes	Yes	Yes	Yes
print(target:Object, boundingBox:String)	Yes	No	No	No

Function Name	ActionScript 2	Flash Lite 2.1	Flash Lite 3.0	Flash Lite 3.1
printAsBitmap(target:Object, boundingBox:String)	Yes	No	No	No
printAsBitmapNum(level:Number, boundingBox:String)	Yes	No	No	No
printNum(level:Number, boundingBox:String)	Yes	No	No	No
random(value:Number)	Deprecated	Deprecated	Deprecated	Deprecated
removeMovieClip(target:Object)	Yes	Yes	Yes	Yes
return	No	Yes	Yes	Yes
set	No	Yes	Yes	Yes
setInterval(functionReference:Function, interval:Number, [param:Object], objectReference:Object, methodName:String)	Yes	Yes	Yes	Yes
setProperty(target:Object, property:Object, expression:Object)	Yes	Yes	Yes	Yes
showRedrawRegions(enable:Boolean, [color:Number])	Yes	No	No	No
startDrag(target:Object, [lock:Boolean], [left,top,right,bottom:Number])	Yes	Yes	Yes	Yes
stop()	Yes	Yes	Yes	Yes
stopAllSounds()	Yes	Yes	Yes	Yes
stopDrag()	Yes	Yes	Yes	Yes
String(expression:Object)	Yes	Yes	Yes	Yes
substring(string:String, index:Number, count:Number)	Deprecated	Deprecated	Deprecated	Deprecated
switch	No	Yes	Yes	Yes
targetPath(targetObject:Object)	Yes	Yes	Yes	Yes
tellTarget(target:String, statement(s))	Deprecated	Deprecated	Deprecated	Deprecated
toggleHighQuality()	Deprecated	Deprecated	Deprecated	Deprecated
trace(expression:Object)	Yes	Yes	Yes	Yes
unescape(string:String)	Yes	Yes	Yes	Yes
unloadMovie(target:Object)	Yes	Yes	Yes	Yes
unloadMovieNum(level:Number)	Yes	Yes	Yes	Yes
updateAfterEvent()	Yes	No	No	No

Unsupported and partially supported classes: details

This chapter describes ActionScript 2.0 classes that are either partially or not supported by Adobe's Macromedia Flash Lite 2.0, Adobe Flash Lite 2.1, and Adobe Flash Lite 3.x. It also describes the extensions that are specific to ActionScript for Flash Lite 2.x and 3.x.

Button class

All button symbols in a SWF file are instances of the Button object. The Button class provides methods, properties, and event handlers for working with buttons.

For more information about the Button class, see the following:

- Chapter 10, “Handling Events,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Property summary

The following table lists the properties that are either partially or not supported by the Button class when using ActionScript for Flash Lite 2.x and 3.x.

Property	Description	Support
menu	An object that associates a ContextMenu object with a button.	Not supported
trackAsMenu	A Boolean value that indicates whether other buttons can receive mouse release events. Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.	Partially supported
useHandCursor	A Boolean value that indicates whether the pointer appears when the mouse passes over a button.	Not supported
_xmouse	Read-only; the x coordinate of the pointer, relative to a button instance. Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.	Partially supported
_ymouse	Read-only; the y coordinate of the pointer, relative to a button instance. Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.	Partially supported

Event summary

The following table lists the event handlers that are partially supported by the Button class when using ActionScript for Flash Lite 2.x and 3.x.

Event	Description
onDragOut	Invoked when the mouse button is clicked over the button and the pointer then dragged outside of the button. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onDragOver	Invoked when the pointer is dragged over the button. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onReleaseOutside	Invoked when the mouse is released while the pointer is outside the button after the button is pressed while the pointer is inside the button. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.

Date class

The Date class shows how dates and times are represented in ActionScript, and it supports operations for manipulating dates and times. The Date class can also obtain the current date and time from the operating system.

For more information about the Date class, see the following:

- Chapter 4, “Data and Data Types,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Method summary

The following table lists the methods (functions) that have been added to the Date class when using ActionScript for Flash Lite 2.x and 3.x.

Method	Description
<code>getLocaleLongDate()</code>	This function sets a parameter to a string that represents the current date, in long form, formatted according to the currently defined locale. The parameter is passed in by name. The returned value is a multiple-character, variable-length string. The actual formatting depends on the device and the locale.
<code>getLocaleShortDate()</code>	This function sets a parameter to a string that represents the current date, in abbreviated form, formatted according to the currently defined locale. The parameter is passed in by name. The returned value is a multiple-character, variable-length string. The actual formatting depends on the device and the locale.
<code>getLocaleTime()</code>	This function sets a parameter to a string that represents the current time, formatted according to the currently defined locale. The parameter is passed in by name. The returned value is a multiple-character, variable-length string. The actual formatting depends on the device and the locale.

Key class

The Key class provides methods and properties for obtaining information about the keyboard itself, and the keypresses input into it.

For more information about the Key class, see the following:

- Chapter 14, “Creating Interaction with ActionScript,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Flash Lite property extensions

The following table lists the property that has been added to the Key class when using ActionScript for Flash Lite 2.x and later.

Property	Description	Support
isToggled	Returns <code>true</code> if the Num Lock or Caps Lock key is active.	Not supported

Flash Lite method extensions

The following table lists the method extension that has been added to the Key class when using ActionScript for Flash Lite 2.x and later.

Method	Description
getCode ()	Returns the virtual key code of the last key pressed.

The Flash Lite 2.x and 3.x implementation of the `getCode ()` method returns a string or a number, depending on what the platform passed in. The only valid key codes are the standard key codes accepted by this class and the “special” key codes listed as properties of the `ExtendedKey` class. This restriction is enforced by the player. For valid key code values, see the Key class in the *Flash Lite 2.x and 3.x ActionScript Language Reference*. This reference provides tables that map keys to codes for letters, numbers, the numeric keypad, function keys, special constant keys, and other keys.

Mouse class

The Mouse class lets you control the mouse in a SWF file; for example, this class lets you hide or show the mouse pointer.

For more information about the Mouse class, see the following references:

- Chapter 14, “Creating Interaction with ActionScript,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Method summary

The following table lists the methods that are either partially or not supported by the Mouse class when using ActionScript for Flash Lite 2.x and later.

Method	Description	Support
<code>addListener ()</code>	Registers an object to receive notifications of the <code>onMouseDown</code> , <code>onMouseMove</code> , <code>onMouseUp</code> , and <code>onMouseWheel</code> listeners. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>hide ()</code>	Hides the pointer in a SWF file.	Not supported
<code>removeListener ()</code>	Removes an object that was previously registered with <code>addListener ()</code> . Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>show ()</code>	Displays the pointer in a SWF file.	Not supported

Event summary

The following table lists the events that are either partially or not supported by the Mouse class when using ActionScript for Flash Lite 2.x and later.

Event	Description	Support
<code>onMouseDown</code>	Notified when the mouse is pressed. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.	Partially supported
<code>onMouseMove</code>	Notified when the mouse moves. Limitations: Supported if <code>System.capabilities.hasMouse</code> is set to true.	Partially supported
<code>onMouseUp</code>	Notified when the mouse is released. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.	Partially supported
<code>onMouseWheel</code>	Notified when the user rolls the mouse wheel.	Not supported

MovieClip class

The MovieClip class lets you use listener callback functions that provide status information while SWF or JPEG files load (download) into movie clips. To use MovieClip features, use `MovieClipLoader.loadClip()` instead of `loadMovie()` or `MovieClip.loadMovie()` to load SWF files.

For more information about the MovieClip class, see the following:

- Chapter 11, “Working with Movie Clips,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Method summary

The following table lists the methods that are either partially or not supported by the MovieClip class when using ActionScript for Flash Lite 2.x and later.

Method	Description	Support
<code>attachAudio()</code>	Captures and plays local audio from the device’s microphone hardware.	Not supported

Method	Description	Support
<code>getTextSnapshot()</code>	Returns a <code>TextSnapshot</code> object that contains the text in the static text fields in the specified movie clip.	Not supported
<code>startDrag()</code>	Specifies a movie clip as draggable and begins dragging the movie clip. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>stopDrag()</code>	Stops a <code>MovieClip.startDrag()</code> method. A movie clip that was made draggable with <code>startDrag()</code> remains draggable until a <code>stopDrag()</code> method is added, or until another movie clip becomes draggable. Only one movie clip is draggable at a time. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported

Property summary

The following table lists the properties that are either partially or not supported by the `MovieClip` class when using ActionScript for Flash Lite 2.x and later.

Property	Description	Support
<code>_droptarget</code>	Returns the absolute path in slash-syntax notation of the movie clip instance on which this movie clip was dropped. The <code>_droptarget</code> property always returns a path that starts with a slash (/). To compare the <code>_droptarget</code> property of an instance to a reference, use the <code>eval()</code> function to convert the returned value from slash syntax to a dot-syntax reference. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>menu</code>	Associates the specified <code>ContextMenu</code> object with the movie clip.	Not supported
<code>_quality</code>	Sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and therefore are unaffected by the <code>_quality</code> property.	Partially supported
<code>trackAsMenu</code>	A Boolean value that indicates whether other buttons or movie clips can receive mouse release events. The <code>trackAsMenu</code> property lets you create menus. You can set the <code>trackAsMenu</code> property on any button or movie clip object. If the <code>trackAsMenu</code> property does not exist, the default behavior is <code>false</code> . Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>useHandCursor</code>	A Boolean value that determines whether the hand icon appears when the mouse rolls over a movie clip.	Not supported
<code>_xmouse</code>	Returns the x coordinate of the mouse position. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>_ymouse</code>	Returns the y coordinate of the mouse position. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported

Event summary

The following table lists the event handlers that are partially supported by the MovieClip class when using ActionScript for Flash Lite 2.x and later.

Event Handler	Description
onDragOut	Invoked when the mouse button is pressed and the pointer rolls outside the object. You must define a function that executes when the event handler is invoked. You can define the function on the timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onDragOver	Invoked when the pointer is dragged outside and then over the movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onMouseDown	Invoked when the left mouse button is pressed. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onMouseMove	Invoked every time the mouse moves. Limitations: Supported if <code>System.capabilities.hasMouse</code> is set to true.
onMouseUp	Invoked every time the left mouse button is pressed. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.
onReleaseOutside	Invoked when the mouse button is pressed over a movie clip and released while the pointer is outside the movie clip's area. Limitations: Supported if <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to true.

NetConnection class

The NetConnection class lets you create an object that you can use with a NetStream object to invoke commands on a remote application server or to play back streaming Flash Video (FLV) files either locally or from a server. Support for this class was added in Flash Lite 3.0.

For more information about the NetConnection class, see the following:

- “Working with Images, Sound, and Video” in *Learning ActionScript 2.0 in Adobe Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

NetStream class

The NetStream class lets you create a stream that can be used with a NetConnection object to play FLV files from a local file system, an HTTP address, or a Flash Media Server. Support for this class was added in Flash Lite 3.0.

For more information about the NetStream class, see the following:

- “Working with Images, Sound, and Video” in *Learning ActionScript 2.0 in Adobe Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Property summary

The following table lists the property that is unsupported by the NetStream class when using ActionScript for Flash Lite 3.0 and later.

Property	Description
checkPolicyFile	Specifies whether the player should attempt to download a cross-domain policy file from the loaded FLV file's server before beginning to load the FLV file itself.

Selection class

The Selection class lets you set and control the text field in which the insertion point is located (that is, the field that has focus).

Partial support for the Selection class was added to support inline text input in Flash Lite 2.1. For Flash Lite, the Selection object is valid only when a device supports inline text entry. If a device does not support inline text entry, and instead relies on an FEP (front-end processor) to enter text, all calls to the Selection object are ignored.

For more information about the Selection class, see Chapter 2, "ActionScript Classes," in *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Method summary

The following table lists the methods that are unsupported by the Selection class when using ActionScript for Flash Lite 2.x and later.

Method	Description
getBeginIndex()	Returns the index at the beginning of the currently focused selection span.
getCaretIndex()	Returns the index of the blinking insertion point (caret) position.
getEndIndex	Returns the index at the end of the currently focused selection span.

SharedObject class

A Flash Lite shared object, as implemented in the ActionScript SharedObject class, allows Flash Lite content to save data to the device when the application is closed and load the data from the device when the application is played again.

For more information about the SharedObject class, see the following references:

- Chapter 14, "Creating Interaction with ActionScript" in *Learning ActionScript 2.0 in Flash*
- Chapter 2, "ActionScript Classes," in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Method summary

The following table lists the methods that are partially supported by the SharedObject class when using ActionScript for Flash Lite 2.x and later.

Method	Description
<code>flush()</code>	Immediately writes a locally persistent shared object to a local file. Limitations: The write operation is asynchronous and the result is not immediately available.
<code>getLocal()</code>	Returns a reference to a locally persistent shared object that is available only to the current client. Limitations: In Flash Lite, a shared object cannot be shared between two SWF files.

Flash Lite method extensions

The following table lists the methods that have been added as extensions to the `SharedObject` class when using ActionScript for Flash Lite 2.x and later.

Method	Description
<code>GetMaxSize()</code>	Flash Lite returns the maximum size allotted for persistent storage of a SWF file.

System class

The `System` class contains properties that are related to certain operations on the user's computer, such as operations with shared objects, local settings for cameras and microphones, and using the Clipboard. The following additional properties and methods are in specific classes within the `System` package: the capabilities class, the security class, and the IME class.

For more information about the `System` class, see the following references:

- *Developing Flash Lite 2.x and 3.x Applications*
- *Flash Lite 2.x and 3.x ActionScript Language Reference*
- Chapter 7, "Classes," in *Learning ActionScript 2.0 in Flash*

Method summary

The following table lists the methods that are not supported by the `System` class when using ActionScript for Flash Lite 2.x and later.

Method	Description	Support
<code>setClipboard()</code>	Replaces the contents of the clipboard with a specified text string.	Not supported
<code>showSettings()</code>	Shows the specified Flash Player Settings panel.	Not supported

Property summary

The following table lists the properties that are not supported by the `System` class for use with ActionScript for Flash Lite 2.x and later.

Property	Description	Support
<code>exactSettings</code>	Specifies whether to use superdomain or exact-domain matching rules when accessing local settings.	Not supported

Event summary

The following table lists the event handlers that are partially supported by the System class for use with ActionScript for Flash Lite 2.x and later.

Event handler	Description	Support
onStatus	Provides a super event handler for certain objects. The SharedObject property is supported. The LocalConnection and NetStream properties are not supported.	Partially supported

capabilities (System.capabilities) class

The System.capabilities class determines the abilities of the system and player hosting a SWF file, which lets you tailor content for different formats. For example, the screen of a cell phone (black and white, 100 square pixels) is different from a 1000-square-pixel color PC screen. To provide appropriate content to as many users as possible, you can use the System.capabilities object to determine the type of device a user has. You can then either specify to the server to send different SWF files based on the device capabilities or tell the SWF file to alter its presentation based on the capabilities of the device.

Property summary

The following table lists the properties that are not supported by the System.capabilites class when using ActionScript for Flash Lite 2.x and later.

Property	Description
hasIME	Indicates whether the system has an input method editor (IME) installed.
manufacturer	A string that indicates the manufacturer of Flash Player, in the format "Adobe OSName" (OSName could be "Windows", "Macintosh", "Linux", or "Other OS Name").
pixelAspectRatio	Indicates the pixel aspect ratio of the screen.
playerType	Indicates the type of player: stand-alone, external, plug-in, or ActiveX.
screenColor	Indicates whether the screen is color, grayscale, or black and white.
screenDPI	Indicates the screen resolution, in pixels (such as dpi).
serverString	A URL-encoded string that specifies values for each System.capabilities property.

Flash Lite property extensions

The following table lists properties that are extensions to the System.capabilites class for use with ActionScript for Flash Lite 2.x and later.

Property	Description
hasCompoundSound	A read-only Boolean value that is true if the player can process compound sound data; false otherwise.
hasEmail	A read-only Boolean value that is true if the player can send e-mail messages using the GetURL ActionScript command; false otherwise.
hasMMS	A read-only Boolean value that is true if the player can send multimedia message service (MMS) messages using the GetURL ActionScript command; false otherwise.

Property	Description
hasSMS	A read-only value whose variable <code>_capsms</code> indicates whether Flash Lite can send short message service (SMS) messages using the <code>GetURL</code> ActionScript command. If Flash Lite can send SMS messages, this variable is defined and has a value of 1. Otherwise, this variable is not defined.
hasMFI	A read-only Boolean value that is <code>true</code> if the player can play sound data in the Melody Format for i-mode (MFI) audio format; <code>false</code> otherwise.
hasMIDI	A read-only Boolean value that is <code>true</code> if the player can play sound data in the MIDI audio format; <code>false</code> otherwise.
hasSMAF	A read-only Boolean value that is <code>true</code> if the player can play sound data in the Synthetic music Mobile Application Format (SMAF); <code>false</code> otherwise.
hasDataLoading	A read-only Boolean value that is <code>true</code> if the player can dynamically load additional data through calls to <code>loadMovie()</code> , <code>loadMovieNum()</code> , <code>loadVariables()</code> , <code>loadVariablesNum()</code> , <code>XML.parseXML()</code> , <code>Sound.loadSound()</code> , <code>MovieClip.loadVariables()</code> , <code>MovieClip.loadMovie()</code> , <code>MovieClipLoader.loadClip()</code> , <code>LoadVars.load()</code> , and <code>LoadVars.sendAndLoad()</code> ; <code>false</code> otherwise.
has4WayKeyAS	A read-only Boolean value that is <code>true</code> if the player can execute ActionScript attached to <code>KeyEvent</code> handlers associated with the Right, Left, Up and Down keys; <code>false</code> otherwise. If the value of this variable is <code>true</code> , when one of the four-way keys is pressed, the player first looks for a handler for that key. If none is found, Flash control navigation is performed. However, if an event handler is found, no navigation action occurs for that key. In other words, the presence of a keypress handler for a Down key disables the ability to navigate down.
hasMouse	A read-only Boolean value that is <code>true</code> if the player can send mouse-related events and <code>false</code> if the platform does not support a mouse.
hasMappableSoftKeys	Allows user to set soft-key values and handle events from those soft keys.
hasStylus	A read-only Boolean value that is <code>true</code> if the player can send stylus-related events and <code>false</code> if the platform does not support a stylus. The <code>onMouseMove</code> event is not supported by a stylus. This flag allows the content to check if this event is supported.
hasCMIDI	A read-only Boolean value that is <code>true</code> if the platform supports CMIDI sound, and <code>false</code> if the platform does not support CMIDI sound.
hasXMLSocket	(Added in Flash Lite 2.1) Indicates whether the host application supports XML sockets.
softKeyCount	A number specifying the number of soft keys that the platform supports.
hasSharedObjects	A read-only Boolean value that is <code>true</code> if Flash content playing back in this application can access Flash Lite shared objects; <code>false</code> otherwise.
hasQWERTYKeyboard	A read-only Boolean value that is <code>true</code> if ActionScript can be attached to all keys found on a standard QWERTY keyboard and the Backspace key; <code>false</code> otherwise.
audioMIMEtypes	A read-only property that contains an array of MIME types that the device's audio codecs support and that can be used by the ActionScript Sound object.

Property	Description
imageMIMEtypes	A read-only property that contains an array of MIME types that the device's image codecs support and that can be used by the <code>loadMovie</code> ActionScript function.
videoMIMEtypes	A read-only property that contains an array of MIME types that the device's video codecs support and that can be used by the ActionScript Video object.
MIMEtypes	A read-only property that contains an array of all the MIME types that the Sound and Video objects support and that can be used by the <code>loadMovie()</code> ActionScript function.

Sound class

ActionScript for Flash Lite 2.x and later supports device sound through the Sound class and through `System.capabilities` values. The Sound class is fully supported for native sounds supported in Flash Player 7, but it is only partially supported for device sounds.

Flash Lite 2.x added support that lets you synchronize device sound playback with rendering animation.

Note: *Flash Lite 2.x and later does not support sound recording.*

Method summary

The following table lists the methods that are partially supported by the Sound class when using ActionScript for Flash Lite 2.x and later.

Method	Description	Support
<code>getPan()</code>	Returns the pan level set in the last <code>setPan()</code> call as an integer from -100 (left) to +100 (right). (0 sets the left and right channels equally.) The pan setting controls the left-right balance of the current and future sounds in a SWF file. Limitations: Supported for use with native Flash sound; not supported for use with device sound.	Partially supported
<code>getTransform()</code>	Returns the sound transform information for the specified Sound object set with the last <code>Sound.setTransform()</code> call. Limitations: Supported for use with native Flash sound; not supported for use with device sound.	Partially supported
<code>loadSound()</code>	Loads an MP3 file into a Sound object. You can use the <code>isStreaming</code> parameter to indicate whether the sound is an event or a streaming sound. Event sounds are completely loaded before they play. They are managed by the ActionScript Sound class and respond to all methods and properties of this class. Limitations: The streaming parameter is ignored when used with Flash Lite 2.x and later.	Partially supported

Method	Description	Support
<code>setPan()</code>	Determines how the sound is played in the left and right channels (speakers). For mono sounds, pan determines which speaker (left or right) the sound plays through. Limitations: Supported for use with native Flash sound; not supported for use with device sound.	Partially supported
<code>setTransform()</code>	Sets the sound transform (or balance) information for a Sound object. The <code>soundTransformObject</code> parameter is an object that you create using the constructor method of the generic Object class, with parameters specifying how the sound is distributed to the left and right channels (speakers). Limitations: Supported for use with native Flash sound; not supported for use with device sound.	Partially supported
<code>setVolume()</code>	Sets the volume for the Sound object. Limitations: Supported for use with native Flash sound; not supported for use with device sound.	Partially supported

Property summary

The following table lists the properties of the Sound class that are partially supported when using ActionScript for Flash Lite 2.x and later.

Property	Description
<code>duration</code>	The duration of a sound, in milliseconds. Limitations: Supported for use with native Flash sound; not supported for use with device sound.
<code>position</code>	The number of milliseconds a sound has been playing. Limitations: Supported for use with native Flash sound; not supported for use with device sound.

Flash Lite method extensions

The following table lists new methods in the Sound class that are specific to ActionScript for Flash Lite 2.x and later.

Method	Description
<code>getPan()</code>	Returns the value of the previous <code>setPan()</code> call. This method is not supported for device sound.
<code>getTransform()</code>	Returns the value of the previous <code>setTransform()</code> call. This method is not supported for device sound.
<code>loadSound()</code>	Loads sound data of any format into Flash Player. This method is different from the Flash Player 7 implementation because sound data loaded using this method is always treated as event sound. Therefore, the second parameter of this method is always ignored. In the following example, the value <code>true</code> is ignored: <code>my_sound.loadSound("mysnd.mp3", true);</code>

Stage class

The Stage class is a top-level class whose methods, properties, and handlers you can access without using a constructor. Use the methods and properties of this class to access and manipulate information about the boundaries of a SWF file.

For more information, see *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Flash Lite property extensions

The following table lists new properties in the Stage class that are specific to ActionScript for Flash Lite 2.x and later.

Property	Description
<code>showMenu</code>	Shows or hides the default items in the Flash Player context menu.
<code>getTransform()</code>	Returns the value of the previous <code>setTransform()</code> call. This method is not supported for device sound.
<code>loadSound()</code>	Loads sound data of any format into Flash Player. This method is different from the Flash Player 7 version because sound data loaded using this method is always treated as event sound, so the second parameter of this method is always ignored. For example, in the following call, the value <code>true</code> is ignored: <code>my_sound.loadSound("mysnd.mp3", true);</code>

TextField class

Text fields are visual elements on the Stage that let you display text to a user. Similar to an input text field or text area form control in HTML, Flash lets you set text fields as editable (read-only), allow HTML formatting, and enable multiline support.

You use the TextField class to create text fields. All dynamic and input text fields in a SWF file are instances of the TextField class. You can give a text field an instance name in the Property inspector and use the methods and properties of the TextField class to manipulate it with ActionScript. TextField instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

To create a text field dynamically, do not use the `NEW` operator; instead, use `MovieClip.createTextField()`.

The methods of the TextField class let you set, select, and manipulate text in a dynamic or input text field that you create during authoring or at runtime.

All properties of the TextField class are supported in Flash Lite 2.x and later, but you can only use text fields to display device fonts. Device fonts are special fonts in Flash that are not embedded in a SWF file. Flash Lite uses whatever font on the mobile device most closely resembles the device font. Because font outlines are not embedded, a SWF file size is smaller than using embedded font outlines. However, because device fonts are not embedded, the text that you create with these fonts looks different than expected on devices that do not have a font installed that corresponds to the device font. Flash includes three device fonts: `_sans` (similar to Helvetica and Arial), `_serif` (similar to Times Roman), and `_typewriter` (similar to Courier).

For more information about the TextField class, see the following references:

- Chapter 12, “Working with Text and Strings,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Property summary

The following table lists the properties of the TextField class that are not supported when using ActionScript for Flash Lite 2.x and later.

Property	Description
antialiasType	Indicates the type of anti-aliasing used for this TextField instance.
menu	Associates a ContextMenu object with a text field
mouseWheelEnabled	A Boolean value that indicates whether Flash Player should automatically scroll multiline text fields when the mouse pointer clicks a text field and the user rolls the mouse wheel.
restrict	Indicates the set of characters a user can enter into the text field.
sharpness	The sharpness of the glyph edges in this TextField instance.
styleSheet	Attaches a style sheet to the text field.
thickness	Indicates the thickness of the glyph edges in this TextField instance.

Method summary

The following table lists the methods of the TextField class that are not supported when using ActionScript for Flash Lite 2.x and later.

Method	Description
getFontList	Returns the names of fonts on the player's host system as an array.

TextFormat class

The TextFormat class represents character formatting information. Use the TextFormat class to create specific text formatting for text fields. You can apply text formatting to static and dynamic text fields. Some properties of the TextFormat class are not available for embedded and device fonts.

The TextFormat class lets you apply formatting to a text field or to certain characters within a text field. Some examples of text formatting options that can be applied to text are alignment, indenting, bold, color, font size, margin widths, italics, and letter spacing. You can apply text formatting to static and dynamic text fields. Some properties of the TextFormat class are not available for embedded and device fonts.

You must use the constructor `TextFormat()` to create a TextFormat object before calling its methods.

Note: Flash Lite 2.x and later provides partial support for the formatting feature available in the TextFormat class. Formatting features are not available when you use device fonts.

Flash Lite 2.x and later provides partial support for the TextFormat class. For example, `TextFormat.font`, `TextFormat.bold`, and `TextFormat.tabstop` are not supported when you use device fonts.

For more information about the TextFormat class, see the following references:

- Chapter 12, “Working with Text and Strings,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Property summary

The following table lists the properties that are partially supported when using ActionScript 2.0 to create Flash content.

Property	Description
<code>bold</code>	A Boolean value that specifies whether the text is boldface. Limitations: For use only with paragraph-level formatting; you cannot apply bold to individual characters.
<code>bullet</code>	A Boolean value that indicates that the text is part of a bulleted list. Limitations: For use only with embedded fonts.
<code>color</code>	Indicates the color of text. Limitations: For use only with paragraph-level formatting; you cannot apply color to individual characters.
<code>font</code>	The name of the font for text in this text format, as a string. Limitations: For Flash Lite, this property works for embedded fonts only. This property is not supported for Arabic, Hebrew, and Thai.
<code>italic</code>	A Boolean value that indicates whether text in this text format is italic. Limitations: For use only with paragraph-level formatting; you cannot apply italics to individual characters.
<code>size</code>	The point size of text in this text format. Limitations: For use only with paragraph-level formatting; you cannot apply different font sizes to individual characters.
<code>tabStops</code>	Specifies custom tab stops as an array of non-negative integers. Limitations: For use with embedded fonts only.
<code>underline</code>	A Boolean value that indicates whether the text that uses this text format is underlined (<code>true</code>) or not (<code>false</code>). Limitations: For use only with paragraph-level formatting.

Video class

Flash Lite 2.x lets you work with device-specific video formats, and supports the following types of video playback:

- Video embedded in a SWF file
- Video available as a separate file on the device
- Video streamed over the network (in real time)

Flash Lite 2.x supports device video. Device video is stored in the published SWF file in the device's native video format. To play the device video, Flash Lite passes the video data to the device, which then decodes and plays the video.

Note: *Flash Lite 2.x ActionScript does not support the `NetConnection` or `NetStream` classes.*

Flash Lite 3.0 adds support for Flash Video (FLV) using versions of the On2 and Sorenson codecs optimized for mobile devices. FLV video is rendered directly in the Flash Lite player rather than by the device, so you no longer need to be concerned about whether your target devices support a particular video format. The ActionScript `NetConnection` and `NetStream` classes, which were not previously available in Flash Lite, let you control the playback of FLV video from a local drive or HTTP address. These classes are described in Chapter 2, "ActionScript Classes," in *Flash Lite 2.x and 3.x ActionScript Language Reference*. Streaming video data over an RTMP connection to a Flash Media Server is also supported in Flash Lite 3.0 (but RTMPT and RTMPS connections are not). The Camera class and recording video are not supported for Flash Video in Flash Lite.

Flash Lite 3.0 also includes a new property in the Video class, `attachVideo`, that specifies a video stream to be displayed within the boundaries of the Video object on the Stage. You use the methods of the NetStream class rather than those of the Video class to control playback of FLV (non-device) video. For example, to pause device video, you use the `Video.pause` method, but to pause FLV video, you use `NetStream.pause`.

For more information about the Video class, see the following references:

- Chapter 15, “Working with images, Sound, and Video,” in *Learning ActionScript 2.0 in Flash*
- Chapter 2, “ActionScript Classes,” in *Flash Lite 2.x and 3.x ActionScript Language Reference*

Method summary

The following table lists the methods that are not supported by the Video class when using ActionScript for Flash Lite 2.x and later.

Method	Description	Support
<code>clear</code>	Clears the image currently displayed in the Video object. This is useful when, for example, you want to display standby information without having to hide the Video object.	Not supported

Property summary

The following table lists the properties of the Video class that are not supported when using ActionScript for Flash Lite 2.x and later.

Property	Description
<code>deblocking</code>	Indicates the type of deblocking filter applied to decoded video as part of postprocessing. Two deblocking filters are available: one in the Sorenson codec and one in the On2 VP6 codec.
<code>height</code>	An integer specifying the height of the video stream, in pixels.
<code>smoothing</code>	Specifies whether the video should be smoothed (interpolated) when it is scaled. For smoothing to work, the player must be in high-quality mode.
<code>width</code>	An integer specifying the width of the video stream, in pixels.

Flash Lite method extensions

The Video class for Flash Lite 2.x and later adds the following new methods.

Method	Description
<code>play()</code>	Opens a video source and begins playing the video.
<code>close()</code>	Stops playing the video, frees the memory associated with this Video object, and clears the Video area onscreen.
<code>stop()</code>	Stops playing the video and continues to render the current frame onscreen. A subsequent call to <code>Video.resume()</code> resumes playing from the first frame of the video.
<code>pause()</code>	Stops playing the video and continues to render the current frame onscreen. A subsequent call to <code>Video.resume()</code> resumes playing from the current position.
<code>resume()</code>	Resumes playing the video.

The Video class for Flash Lite 3.0 added the following new method.

Method	Description
<code>attachVideo()</code>	Specifies a video stream (source) to be displayed within the boundaries of the Video object on the Stage.

Unsupported and partially supported ActionScript elements: details

This section describes the global functions and properties, constants, operators, statements, extensions, and keywords that are either partially supported or not supported by ActionScript for Macromedia Flash Lite 2.x software from Adobe, or by Adobe Flash Lite 3.x.

Commands issued through `fsCommand` and `fsCommand2`

The `fsCommand()` and `fsCommand2()` global functions let the SWF file communicate with either Flash Player or the program that is hosting Flash Player, such as a web browser.

Flash Lite 2.x and later modifies the standard Macromedia Flash Player 7 commands, and adds commands that are specific to embedded devices. Flash Lite 2.x and later also supports the `fsCommand2()` function, which provides similar functionality to `fsCommand()`, with the exception that the command is executed immediately, not deferred until after the calling frame is processed.

For more information on the `fsCommand2()` function, see the *Flash Lite 2.x and 3.x ActionScript Language Reference*.

Unsupported commands

The following table lists the commands that are not supported by `fsCommand()` when using ActionScript 2.0 to create Flash Lite content.

Command	Description
<code>quit</code>	Closes the projector.
<code>fullscreen</code>	Specifying <code>true</code> sets Flash Player to full-screen mode. Specifying <code>false</code> returns the player to normal menu view.
<code>allowscale</code>	Specifying <code>false</code> sets the player so the SWF file is always drawn at its original size and is never scaled. Specifying <code>true</code> forces the SWF file to scale to 100% of the screen.
<code>showmenu</code>	Specifying <code>true</code> enables the full set of context menu items. Specifying <code>false</code> hides all the context menu items except About Flash Player and Settings.
<code>exec</code>	Executes an application from within the projector.
<code>trapallkeys</code>	Specifying <code>true</code> sends all key events, including accelerator keys, to the <code>onClipEvent(keyDown/keyUp)</code> handler in Flash Player.

Commands summary

The following table lists the ActionScript commands, functions, and keywords that are partially or not supported by Flash Lite 2.x and later.

Command, function, or keyword	Description	Support
<code>asfunction</code>	A protocol for URLs in HTML text fields.	Not supported
<code>fscommand()</code>	Function that lets the SWF file communicate with either Flash Player or the program hosting Flash Player, such as a web browser.	Partially supported
<code>on</code>	Prefix for events to execute when the event occurs. Limitation: The supported events are <code>rollout</code> , <code>rollover</code> , and <code>keyPress</code> .	Partially supported
<code>onClipEvent</code>	Event handler; triggers actions defined for a specific instance of a movie clip. Limitation: The Supported events are <code>press</code> , <code>load</code> , <code>unload</code> , <code>enterFrame</code> , <code>keyDown</code> , <code>keyUp</code> , and <code>data</code> . The <code>mouseDown</code> , <code>mouseUp</code> , and <code>mouseMove</code> events are supported if either <code>System.capabilities.hasMouse</code> or <code>System.capabilities.hasStylus</code> is set to <code>true</code> .	Partially supported
<code>onUpdate</code>	The <code>onUpdate</code> event handler is defined for a live preview used with a component.	Not supported
<code>print()</code>	The <code>print()</code> function targets the movie clip according to the boundaries specified in the parameter.	Not supported
<code>printAsBitmap()</code>	A function that prints the target movie clip as a bitmap.	Not supported
<code>printAsBitmapNum()</code>	A function that prints a level in Flash Player as a bitmap according to the boundaries specified in the parameter (<code>bmovie</code> , <code>bmax</code> , or <code>bframe</code>).	Not supported
<code>printNum()</code>	A function that prints the level in Flash Player according to the boundaries specified in the <code>boundingBox</code> parameter (<code>bmovie</code> , <code>bmax</code> , or <code>bframe</code>).	Not supported
<code>startDrag()</code>	A function that makes the target movie clip draggable while the SWF is playing. Only one movie clip can be dragged at a time. Limitation: Supported if mouse or stylus interface is supported.	Partially supported
<code>stopDrag()</code>	A function that stops the current drag operation. Limitation: Supported if mouse or stylus interface is supported.	Partially supported
<code>updateAfterEvent()</code>	A function that updates the display (independent of the frames per second set for the SWF file) when you call it in an <code>onClipEvent()</code> handler or as part of a function or method that you pass to <code>setInterval()</code> .	Not supported

Global properties

The following table lists the ActionScript global properties that are partially supported or not supported by Flash Lite 2.x and later.

Properties	Description	Support
<code>_droptarget</code>	Read-only property that returns the absolute path in slash (/) syntax notation of the movie clip instance on which <code>draggableInstanceName</code> (the name of a movie clip instance that was the target of a <code>startDrag()</code> function) was dropped. This property always returns a path that starts with a slash (/).	Not supported
<code>_highquality</code>	Global property that specifies the level of anti-aliasing applied to the current SWF file. This property can also control bitmap smoothing. Limitation: Flash Lite does not support bitmap smoothing.	Partially supported
<code>_url</code>	Read-only property that retrieves the URL of the SWF file from which the movie clip was downloaded.	Not supported

Chapter 11: Warning and error messages

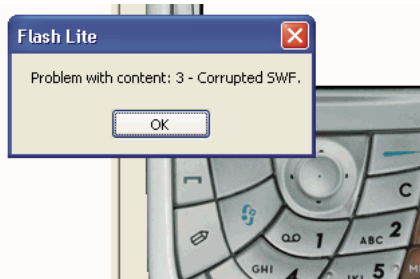
This appendix lists the possible information and warning messages that both the device and the Adobe Device Central emulator might generate while you're testing an Adobe Flash Lite application.

Device error codes

The Adobe Device Central emulator can generate alerts as you test your content. One type of alert appears only in the emulator and is intended to provide information about actual or potential errors; the other type of alert occurs in both the emulator and on an actual device.

The first type of alert provides debugging information about your SWF file. For example, if your SWF file contains Adobe ActionScript that isn't supported by Flash Lite (or by the version of Flash Lite available on the currently selected test device), the emulator generates an alert.

The other type of message that can occur in the emulator also occurs on an actual device. These types of errors are displayed in an error dialog box that the user must close for the application to continue. The following image shows an example error dialog box as it appears in the emulator:



On a device, the error dialog box that appears contains the string “Problem with content” followed by an error number. In the emulator, the error dialog box also contains a short error string. The emulator also displays a longer description of the error in the Output panel.

The following table lists all of the errors that occur in the Flash Lite player, including error numbers, the short descriptions that appear in the error dialog box, and the longer descriptions that appear in the Output panel.

Error number	Error string	Description and possible causes
1	Out of memory	The emulator has run out of heap memory. Unless otherwise specified, the emulator allocates 1 MB of memory for a SWF file to use.
2	Stack limit reached	The emulator has detected that its stack limit is reached or exceeded. This could be caused by various reasons, including multiple levels of nested movie clips or complicated vector drawings.
3	Corrupted SWF	The emulator has detected that the SWF file data is corrupted.

Error number	Error string	Description and possible causes
4	ActionScript stuck	The emulator has detected that certain ActionScript code in the SWF file is taking too long to execute. As a result, the emulator has stopped executing the ActionScript code.
5	N/A	N/A
6	Bad image data	The SWF file contains an image that either Flash Lite, or the platform's native image decoder, was unable to decode.
7	Bad sound data	The SWF file attempted to load a sound in an unsupported format, or the sound data is corrupted.
8	Root movie unloaded	This error occurs when the root (level 0) SWF file is replaced with another SWF file.
9	N/A	N/A
10	getURL string too long	The URL string in the getURL call is too long.
11	Not enough memory to decompress image.	Flash Lite does not have enough memory to decode an image in the SWF file.
12	Bad SVG data	Flash Lite attempted to load SVG data that is corrupted.
13	Stream loading out of memory	Flash Lite does not have enough memory available to handle the data being streamed from a URL. For example, this error can occur if you attempt to load an XML file over the network that's too large for Flash Lite to handle at one time. If possible, try breaking the data file into several smaller files and then load each file individually.

Adobe Device Central emulator error and information messages

Messages appear in a pop-up Output window in the emulator and Adobe Flash also reports them in its Output panel. The following table lists all the information messages that the Adobe Device Central emulator reports:

Error code	Message	Description
FTPA002	FSCommand is ignored.	The emulator detected an <code>fscommand()</code> function call, which is not supported by the selected test device. No modifications are made to the device-specific SWF file—this is just a warning.
FTPA003	loadVariables is ignored.	The emulator detected a <code>loadVariables()</code> function call, which is not supported by the selected test device and content type. No modifications are made to the device-specific SWF file—this is just a warning.
FTPA004	loadMovie is ignored.	The emulator detected a <code>loadMovie()</code> function call, which is not supported by the selected test device and content type. No modifications are made to the device-specific SWF file—this is just a warning.

Error code	Message	Description
FTPA005	The call to GetURL for <i>URL</i> was ignored because there was more than one request per keypress.	Flash Lite allows only one <code>getURL()</code> function call per keypress; the emulator detected that there was more than one <code>getURL()</code> so only the first command is processed—the others are ignored.
FTPA006	The call to GetURL for <i>URL</i> was ignored because it was not associated with a keypress.	The currently selected test device and content type process only <code>getURL()</code> function calls that result from users pressing a key on their device. The emulator detected that your application made a call to <code>getURL()</code> that wasn't associated with a keypress.
FTPA007	<code>getProperty</code> or <code>setProperty</code> not supported for: <i>movieclip</i> property.	Flash Lite does not support the specified movie clip property.
FTPA008	<code>getProperty</code> or <code>setProperty</code> not fully supported for: <i>movieclip</i> property.	Flash Lite does not fully support the specified movie clip property. For more information, see the entry for the specified property in the <i>Flash Lite 2.x and 3.0 ActionScript Language Reference</i> .
FTPA009	<code>startDrag</code> and <code>stopDrag</code> are not supported.	The emulator detected a <code>startDrag()</code> or <code>stopDrag()</code> function call, which Flash Lite does not support.
FTPA014	<code>getURL</code> is ignored.	The emulator detected a <code>getURL()</code> function call, which is not supported by the selected test device and content type. No modifications are made to the device-specific SWF file—this is just a warning.
FTPA015	The call to loadMovie for <i>URL</i> was ignored because there was more than one request per keypress.	Flash Lite allows only one <code>loadMovie()</code> function call per keypress; the emulator detected that there was more than one <code>loadMovie()</code> so only the first command is processed—the others are ignored.
FTPA016	The call to loadMovie for <i>URL</i> was ignored because it was not associated with a keypress.	The currently selected test device and content type process only <code>loadMovie()</code> function calls that result from users pressing a key on their device. The emulator detected that your application made a call to <code>loadMovie()</code> that wasn't associated with a keypress.
FTPA017	The call to loadVariables for <i>URL</i> was ignored because there was more than one request per keypress.	Your application made multiple <code>loadVariables()</code> function calls during a single keypress event. Flash Lite allows only one <code>loadVariables()</code> command per keypress, so only the first command is processed—the others are ignored.
FTPA018	The call to loadVariables for <i>URL</i> was ignored because it was not associated with a keypress.	The currently selected test device and content type process only <code>loadVariables()</code> function calls that result from users pressing a key on their device. The emulator detected that your application made a call to <code>loadVariables()</code> that wasn't associated with a keypress.
FTPA019	The call to FSCommand with arguments <i>command-arguments</i> was ignored because there was more than one request per keypress.	Flash Lite allows only one <code>fscommand()</code> function call per keypress; the emulator detected that there was more than one <code>fscommand()</code> so only the first command is processed—the others are ignored.

Error code	Message	Description
FTPA020	The call to <code>FSCCommand</code> with arguments <i>command-arguments</i> was ignored because it was not associated with a keypress.	The currently selected test device and content type process only <code>fscCommand()</code> function calls that result from users pressing a key on their device. The emulator detected that your application made a call to <code>fscCommand()</code> that wasn't associated with a keypress.
FTPE001	The key will not be processed: <i>keyname</i> ASCII Value: <i>value</i> .	The emulator detected that a device key was pressed that isn't supported by Flash Lite—the keypress is ignored.
FTPE013	Input text fields are not supported for the selected content type on this device.	The current test device and content type do not support input text fields.
FTPS010	Streaming Sound is unsupported.	The selected test device and content type do not support streaming sound.
FTPS011	Only a single sound can be played at a time (no mixing).	The emulator detected that the SWF file contains multiple sounds playing simultaneously, which is not supported in Flash Lite.
FTPS012	Event sound was ignored because it was not associated with a keypress.	In Flash Lite 1.0, a sound can play only in response to users pressing a key on their device. (This restriction does not apply to Flash Lite 2.0 and later.)
FTPS021	Sound not supported for the selected content type on this device.	The selected test device and content type do not support sound.
FTPS022	ADPCM sounds not supported for the selected content type on this device.	The emulator detected that the SWF file contains a native (non-device) sound compressed with ADPCM compression, which is not supported by the selected content type on this device. No modifications are made to the device-specific SWF file—this is just a warning.
FTPS023	MP3 sounds not supported for the selected content type on this device.	The emulator detected that the SWF file contains a native (non-device) sound compressed with mp3 compression, which is not supported by the selected content type on this device. No modifications are made to the device-specific SWF file—this is just a warning.
FTPS024	MIDI sounds not supported for the selected content type on this device.	The emulator detected a MIDI device sound, which is not supported by the selected content type on this device.
FTPS025	PCM sounds not supported for the selected content type on this device.	The emulator detected a native Flash sound compressed using PCM compression, which is not supported by the selected content type on this device.
FTPS026	Debug movie is not supported in the specified test movie player.	The Adobe Device Central emulator does not support the Control > Debug Movie menu command.
FTPS027	Sound Bundle found.	The emulator detected that the SWF file contains a sound bundle file.

Error code	Message	Description
FTPS028	Invalid FSCOMMAND2 <i>command-name</i> command found.	The specified <code>fscommand2()</code> command is not a valid command string. For a list of valid <code>fscommand2()</code> commands, see <code>fscommand2</code> function in the <i>Flash Lite 2.x and 3.0 ActionScript Language Reference</i> .
FTPS029	FSCOMMAND2 <i>command-name</i> command found.	The emulator detected the specified <code>fscommand2()</code> command.
FTPS030	FSCOMMAND2 <i>command-name</i> command not supported in the emulator, please test it on the device.	The emulator does not support the specified <code>fscommand2()</code> command. You need to test this SWF file on a device with Flash Lite installed to see whether the specified command functions as expected.
FTPS031	More than one instance of URL Request calls found, only one allowed per keypress/frame.	Flash Lite allows only one <code>getURL()</code> function call per keypress or frame; the emulator detected more than one <code>getURL()</code> so only the first command is processed—the others are ignored.
FTPS032	A call to <code>GetURL(URL)</code> found, limitations might apply.	The emulator detected a <code>getURL()</code> function call, which may have some run-time restrictions when played on the selected device. Test your SWF file on an actual device to see whether the command functions as expected.
FTPS033	A call to <code>loadVariables(URL)</code> found, limitations might apply.	The emulator detected a <code>loadVariables()</code> function call, which may have some run-time restrictions when played on the selected device. Test your SWF file on a device to ensure that the command functions as expected.
FTPS034	A Call to <code>FSCOMMAND (command-name)</code> found, limitations might apply.	This is just a warning that not all devices and Flash Lite content types may support the <code>fscommand()</code> in the application. Test your SWF file on a device to ensure that the command functions as expected.
FTPS035	A call to <code>loadMovie(URL)</code> found, limitations might apply.	The emulator detected a <code>loadMovie()</code> function call, which may have some run-time restrictions when played on the selected device. Test your SWF file on a device to ensure that the command functions as expected.
FTPS036	<i>N</i> kilobytes of <i>device-sound</i> sound found in sound bundle.	For each sound in a sound bundle, the emulator reports the type (for example, MIDI or SMAF) and size of each sound in the bundle.
FTPS037	SMAF sounds not supported for the selected content type on this device.	The emulator detected a SMAF device sound, which is not supported by the selected content type on this device.
FTPS038	The call to <code>StartVibrate</code> was ignored because there was more than one request per frame or event.	Flash Lite allows only one <code>fscommand2("StartVibrate")</code> call per keypress or frame; the emulator detected more than one, so only the first command is processed—the others are ignored.
FTPS039	FSCOMMAND2 <code>SetInputTextType (command-arguments)</code> found, not supported in the emulator, please test it on the device.	The <code>SetInputTextType</code> command is not supported in the emulator. You must test it on an actual device.

Error code	Message	Description
FTPS048	Four Way Navigation is not supported for this device.	The currently selected test device and content type supports two-way navigation. You pressed the left or right arrow keys on the emulator's five-way keypad, which aren't supported in two-way navigation. For more information, see "Default navigation modes" on page 35.
FTPS049	Four Way Navigation with wraparound is not supported for this device.	The currently selected test device and content type supports four-way navigation. You pressed one of the device's arrow keys when there were no objects on the Stage to receive focus in the direction of the arrow key that you pressed. For more information, see "Default navigation modes" on page 35.
FTPS050	Generic MFi sounds not supported for the selected content type on this device.	The emulator detected a Generic MFi device sound, which is not supported by the selected content type on this device.
FTPS051	Unsupported Mouse Event (<i>event-name</i>) found.	The specified mouse event is not supported by the selected test device and content type.
FTPS067	SMAF(MA-2) sounds not supported for the selected content type on this device.	The emulator detected a SMAF (MA-2) device sound, which is not supported by the selected content type on this device.
FTPS068	SMAF (MA-3) sounds not supported for the selected content type on this device.	The emulator detected a SMAF (MA-3) device sound, which is not supported by the selected content type on this device.
FTPS069	SMAF (MA-5) sounds not supported for the selected content type on this device.	The emulator detected a SMAF (MA-5) device sound, which is not supported by the selected content type on this device.
FTPS070	MFi sounds with Fujitsu extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with a Fujitsu extension, which is not supported by the selected content type on this device.
FTPS071	MFi sounds with Mitsubishi extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with a Mitsubishi extension, which is not supported by the selected content type on this device.
FTPS072	MFi sounds with NEC extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with an NEC extension, which is not supported by the selected content type on this device.
FTPS073	MFi sounds with Panasonic extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with a Panasonic extension, which is not supported by the selected content type on this device.
FTPS074	MFi sounds with Sharp extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with a Sharp extension, which is not supported by the selected content type on this device.
FTPS075	MFi sounds with Sony extension not supported for the selected content type on this device.	The emulator detected an MFi device sound with a Sony extension, which is not supported by the selected content type on this device.

Error code	Message	Description
FTPS099	Print commands are not supported.	Your application contains one of the ActionScript print commands (for example, <code>print()</code> or <code>printAsBitmap()</code>), which are not supported by Flash Lite.
FTPS100	<i>Device-sound</i> sound is chosen in sound bundle.	Indicates the name of the device sound in a sound bundle that the emulator played.
FTPS101	None of the formats in the sound bundle are supported on this device.	Indicates that none of the device sounds in a sound bundle are supported by the selected content type on this device.
FTPS102	SMAF sound playback failed.	The emulator was not able to play the SMAF sound.
FTPS105	This SWF is not in Flash Lite format.	Your application attempted to load a SWF file whose version was not in the Flash Lite format; Flash Lite can load other Flash Lite SWF files or Flash 4-formatted SWF files only.
FTPS106	Mouse Event (<i>event-name</i>) was ignored because it was not triggered by Keypress.	The emulator detected a mouse event over a button in your Flash Lite application. The current test device does not support a stylus or touchscreen interface, so you can only interact with buttons on the screen using the emulator's keypad or equivalent keyboard shortcuts.
FTPS107	The key will not be processed: <i>device-key</i> . Use <code>FSCommand2 SetSoftKeys</code> to enable this key.	You pressed one of the soft keys on the emulator's keypad without first calling the <code>SetSoftKeys</code> command. For more information, see "Using the soft keys" on page 47.
FTPS108	Invalid <code>FSCommand</code> (<i>command-name</i>) found.	The specified <code>fscommand()</code> command is not a valid command string.
FTPS109	<code>FSCommand</code> (<i>command-name</i>) not supported in the emulator, please test it on the device.	The emulator does not support the specified <code>fscommand()</code> command. Test this SWF file on a device with Flash Lite installed to see whether the specified command functions as expected.
FTPS110	Soft keys are not supported in the Flash Lite 1.0 player.	The emulator detected that you pressed one of its soft keys but your document's SWF file's version publish setting is set to Flash Lite 1.0. Flash Lite 1.0 does not support soft keys.

Index

A

ActionScript
 controlling device video with 81
 controlling Flash video with 72
 ActionScript FS commands (BREW) 84
 Adobe Device Central emulator
 debug options 103
 defined 1, 102
 versus BREW Simulator 87
 ADS, defined 82
 animated ring tones (BREW) 84
 Application Download Server (*see* ADS)
 application modes, in Flash Lite 104
 applications (BREW)
 specifying as content type 94
 targeting for development 87
 AppLoader tool, tips 98
 available Stage size 14

B

BAR files, defined 92
 BDS, defined 82
 Binary Runtime Environment for Wireless (*see* BREW)
 BREW
 file types 89
 uploading files to devices 97
 BREW 2.x
 device file structure 92
 uploading files to devices 99
 BREW 3.x
 device file structure 92
 uploading files to devices 100
 BREW Delivery System (*see* BDS)
 BREW devices, testing applications on 101
 BREW SDK
 installing 86
 versions supported 85
 BREW Simulator
 versus Adobe Device Central emulator 87
 BREW Tools Suite, installing 86
 browsing, web content 8

Button class
 unsupported or partially supported event handlers 155
 unsupported or partially supported properties 155
 button events 40

C

Cafe Townsend application
 creating navigation 22
 creating the specials animation 20
 content management 9
 content types 16
 content types (BREW)
 applications 87
 screensavers 87
 content types in Flash Lite, described 104
 creating navigation
 creating a key catcher button 22
 using the soft keys 22

D

data, persistent (BREW) 83
 Date class
 unsupported or partially supported methods 156
 determining supported audio file formats 67
 device file structure
 BREW 2.x 92
 BREW 3.x 92
 device pack (BREW)
 downloading 89, 96
 generic vs. specific 89, 96
 device sound 67
 _forceframerate property 67
 synchronizing with animation 67
 device video
 about 74
 bundled 75
 controlling with ActionScript 81
 importing 75
 in emulator 107
 device video, limitations 74
 devices (BREW)
 file structure for BREW 2.x 92
 file structure for BREW 3.x 92

supported 85
 targeting for development 87

DLL files
 defined 89
 naming conventions 90
 drivers (USB), installing 86
 dynamic text fields 22

E

embedded Flash video 72
 embedding font outlines, about 60
 emulator versus simulator (BREW) 87
 Extension, installing on devices (BREW) 98

F

file types
 BREW 89
 icons 95
 naming conventions 90
 Flash Lite
 Adobe Device Central emulator 14
 authoring overview 15
 content types 16
 Flash Lite 2.1 for BREW
 defined 82
 Flash Lite emulator
 features unsupported by 102
 interacting with 103
 warning and error messages 174
 Flash Lite Publisher for BREW
 defined 82
 Flash Media Server 11
 Flash Player 4 and Flash Player 7
 differences between 108
 Flash video
 about 71
 controlling with ActionScript 72
 embedded 72
 font outlines, embedding in SWF files 60
 font rendering methods, applying to text fields 58
 fsCommand() and fsCommand2() 171
 partially supported commands 171
 unsupported commands 171

G

getURL() (BREW)
 about 84
 limitations 88
 global properties
 partially supported 172

H

hardware requirements (BREW) 83
 Hello World application 14

I

Identify Applet screen (wizard) 94
 Include Applet Information screen (wizard) 94
 inline text (BREW) 83
 input text fields
 example application 57
 restricting characters in 56
 interactivity
 creating a menu with buttons 43
 creating, with buttons 40
 detecting keypresses 46
 handling keypress events 40
 tab navigation 35

K

Key class
 new methods 157
 new properties 157
 keypress events
 ActionScript key codes 34
 creating a key listener 46
 handling with ActionScript 40
 supported keys 33
 writing a key handler script 34

L

limitations
 device video 74
 limitations (BREW)
 getURL() 88
 loadVars() 88
 XMLSend() 88

M

mask layers 20
 media, shared (BREW) 93
 menus, creating with buttons 43

MIF files
 defined 89
 naming conventions 90
 MMS 84
 MOD files
 defined 89
 naming conventions 90
 Mouse class
 unsupported or partially supported events 158
 unsupported or partially supported methods 157
 MovieClip class
 partially supported event handlers 160
 unsupported or partially supported methods 158

N

naming conventions, file types (BREW) 90
 National Software Testing Laboratories (*see* NSTL)
 native sound
 about 67
 re-sampling at 8kHz 68
 navigation. *See* tab navigation
 NetConnection
 using with Flash video 71
 NetStream
 using with Flash video 71
 NSTL, defined 82

P

persistent data (BREW) 83
 persistent data (non-BREW) 12
 Publish Settings (BREW)
 post-processor list 93

R

registering as a BREW developer 86
 rendering quality, default quality 59
 requirements (BREW)
 hardware 83
 software 83

S

screensavers (BREW)
 specifying as content type 94
 targeting for development 87
 scrolling text 61

SDK (BREW)
 installing 86
 versions supported 85
 security
 new Flash 3.0 security model 8
 sandboxes 9
 Selection class 161
 unsupported methods 161
 shared media (BREW) 93
 SharedObject class
 new methods 162
 partially supported methods 161
 SIG files
 defined 89
 naming conventions 90
 obtaining 92
 simulator versus emulator (BREW) 87
 sizes, icons 95
 SMS 84
 soft keys 22
 software requirements (BREW) 83
 sound
 bundled 63
 device 63
 external 66
 native 67
 sound (BREW)
 decoding 84
 streaming (BREW) 84
 Sound class
 new methods 166
 partially supported methods 165
 partially supported properties 166
 Stage class
 new properties 167
 Stage, screen size and available 106
 streaming (BREW)
 sound 84
 video 84
 Summary of Your Suggestions screen (wizard) 95
 supported (BREW)
 devices 85
 SDK versions 85
 System class
 partially supported events 163
 unsupported methods 162
 unsupported properties 162

System.capabilities class
 new properties 163
 unsupported properties 163

T

tab navigation
 about 35
 example application using 43
 focus rectangle 37
 four-way with wrap-around 36
 guidelines for 38
 modes of 35
 target devices 16
 targeting, devices for development (BREW) 87
 TBT, defined 82
 test devices 16
 testing (BREW)
 on BREW devices 101
 simulator versus emulator 87
 text fields
 creating scrolling text 61
 dynamic 22
 input text fields, using 51
 rendering quality 59
 restricting characters in input text fields 56
 setting properties 22
 text, inline (BREW) 83
 TextField class
 unsupported methods 168
 unsupported properties 167
 TextFormat class
 partially supported properties 168
 tips, for using AppLoader tool 98
 Tools Suite (BREW), installing 86
 True BREW Testing (*see* TBT)
 tweened animation 20

U

uploading files to BREW devices
 BREW 2.x 99
 BREW 3.x 100
 workflow 98
 USB drivers, installing (BREW) 86

V

video
 device 74
 Flash (FLV) 71
 streaming (BREW) 84
 Video class
 new methods 170
 unsupported methods 170
 unsupported properties 170

W

wallpaper (BREW) 84
 web content, browsing 8
 wizard (BREW)
 Applet Icon screen 95
 Define Output Settings screen 95
 Identify Applet screen 94
 Include Applet Information screen 94
 Summary of Your Suggestions screen 95
 workflow for authoring content 15

X

XML (BREW)
 data handling 83
 sockets 83
 XML.Send(), limitations (BREW) 88
 XMLSocket 11